UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

OBOGATENE SPLETNE APLIKACIJE, AJAX IN ORODJARNE JAVASCRIPT

DIPLOMSKO DELO VISOKOŠOLSKEGA STROKOVNEGA ŠTUDIJA

Bojan Ličen

Ljubljana, 2009

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00467/2009
Datum: 01.09.2009

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **BOJAN LIČEN**

Naslov: **OBOGATENE SPLETNE APLIKACIJE, AJAX IN ORODJARNE JAVASCRIPT**

RICH INTERNET APPLICATIONS, AJAX AND JAVASCRIPT TOOLKITS

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija

Tematika naloge:

Opišite trende v razvoju spletnih aplikacij in primerjajte različne načine njihove izvedbo. Pri tem se posebej posvetite tehniki AJAX, njeni zgodovini in načinu uporabe.

Oglejte si različne orodjarne in ogrodja za delo z JavaScriptom, ki olajšajo razvoj obogatenih spletnih aplikacij. Izberite dva primera takšnih orodjarn in jih podrobneje analizirajte ter pokažite primer njune uporabe.

Mentor:

doc. dr. Janez Demšar

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Franc Solina

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

OBOGATENE SPLETNE APLIKACIJE, AJAX IN ORODJARNE JAVASCRIPT

DIPLOMSKO DELO NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Bojan Ličen

Mentor: doc. dr. Janez Demšar

Ljubljana, 2009

ZAHVALA

Iskreno se zahvaljujem mentorju doc. dr. Janez Demšarju za vodenje in strokovno pomoč ter za ves vložen trud, brez katerega bi bila ta naloga pomanjkljivejša. Zahvalil bi se tudi staršem, ki so me v veh letih študija podpirali in mi tudi vse skupaj omogočili.

VSEBINA

KRATICE

Kratica	Pomen	Razlaga
RIA	Rich internet aplication	Obogatene spletne aplikacije
API	Application programming interface	Vmesnik do programske kode
HTML	Hyper text markup language	Označevalni jezik
XML	Extesible Markup Language	Razširljiv označevalni jezik
AIR	Adobe integrated runtime	Adobovo integrirano okolje
JVM	Java virtual machine	Javanski virtualni stroj
IDE	Integrated development enviroment	Integrirano razvojno okolje
URL	Uniform resource locators	Enolični krajevnik vira
CSS	Cascade style sheets	Prekrivni slogi
MVC	Model, view, controller	Model, pogled, kontrolnik
JSON	Javascript object notation	Javascript objektna notacija
OOP	Object oriented programming	Objektno usmerjeno programiranje

POVZETEK

Število tehnologij, ki nam omogočajo izdelavo obogatenih spletnih aplikacij, v zadnjem času hitro narašča. Razvoj na tem področju je napredoval. Tako se soočamo z najrazličnejšimi pristopi in metodami, ki jih lahko uporabimo. Pravilno izbrati ni vedno lahko, saj je to povezano s sprejemanjem kompromisnih rešitev.

V prvem delu preučuje naloga preučuje nekatere tehnologije, ki so se uveljavile kot primerne rešitve za izdelavo obogatenih spletnih aplikacij. Razvrščanje v kategorije glede na platformo, na kateri se izvajajo, izpostavlja njihove prednosti in slabosti uporabe.

Drugi del naloge podrobno predstavlja pristop Ajax in z njim povezane tehnologije, ki se v povezavi z orodjarnami Javascript pojavljajo kot možna rešitev pri izdelavi obogatenih spletnih aplikacij.

Tretji del naloge najprej analizira lastnosti orodjarn Javascript na splošno. V nadaljevanju sledi primerjava dveh pogosto uporabljenih orodjarn Javascript ter razlik v njunih pristopih in načinih uporabe, s katerimi se srečamo pri razvoju spletnih aplikacij.

Predstavljene tehnologije so v razvoju današnjih spletnih aplikacij vedno bolj prisotne, ker omogočajo izdelavo naprednejših, bolj odzivnih in uporabniku prijaznejših spletnih aplikacij. Pred pričetkom projekta je potrebno preučiti obstoječe tehnologije, saj je njihova ustrezna izbira in vedenje o tem, kaj omogočajo, lahko ključnega pomena pri uspešni izvedbi projekta.

Ključne besede: obogatene spletene aplikacije, orodjarne Javascript, Ajax, interaktivne aplikacije.

ABSTRACT

The number of technologies that enable manufacturing rich web applications is growing rapidly recently. We are facing a wide range of different approaches and ways of implementation that that can be used. Choosing the right one is not always easy task. And it is almost certainly related to the adoption of compromise solutions.

First part of this thesis examines some technologies that have established itself as a viable solutions for production of rich web applications in last few years. Their classification into categories, depending on the platform on which are implemented, pointing out strengths and weaknesses of their usage.

The second part specifically examines the Ajax approach, which appears as a possible solution together with the Javascript toolkits for manufacturing rich web applications.

The third part firstly examines characteristics of the Javascript toolkits in general. Followed by comparison between two most frequently used Javascript toolkits presenting the differences in their approaches and modalities with which are encountered in the development of web applications.

Introduced technologies are increasingly present at today's web applications development. They are enabling advanced, more responsive and user-friendly implementation of web applications. Selection of the appropriate technology must be made before starting the project. Knowing what each technology provides, may be crucial for successfully implemented project.

Key words: rich internet applications, Javascript toolkits, Ajax, interactive applications

1 UVOD

Začetek nove ere svetovnega spleta, poimenovanega s terminom »*Web 2.0*«, je prinesel številne priložnosti. Čeprav je medmrežje staro že skoraj štiri desetletja, je njegova rast in priljubljenost največja prav v obdobju, ko smo stopili v novo tisočletje. Tako imenovana naslednja generacija svetovnega spleta ne predstavlja le novega načina dojemanja in uporabe medmrežja (socialna omrežja, »*wiki-ji*«, spletni dnevniki, itd.), kakršnega smo poznali v pretekli oziroma prvi generaciji svetovnega spleta (zanj se uporablja termin »*Web 1.0*«), temveč postavlja v ospredje predvsem uporabnika – ne le kot ustvarjalca vsebin, ampak tudi tistega, ki vsebine preureja, posodablja, ocenjuje, jih deli z drugimi uporabniki. Tako zasnovan model, za katerega so najpomembnejši vidik uporabniki in njihovi prispevki skupnosti, poimenujemo arhitektura sodelovanja [05]. Ta je temelj, na katerem sloni celoten koncept druge generacije svetovnega spleta. Poleg tega se uporabljajo tudi druge, bolj ali manj inovativne, tehnike in koncepti. Ena izmed njih je zagotovo Ajax. Ta predstavlja skupek tehnologij (programski jezik Javascript, asinhrono izvajanje zahtev, *XML*), ki so v svetovnem spletu poznane že od poznih devetdesetih let. Čeprav so bile vse tehnologije, ki jih združuje nov pristop Ajax (je okrajšava za asinhroni Javascript in *XML*), poznane že od prej, je ta začel pridobivati na pomenu šele v letu 2005, ko ga je v članku predstavil in uvedel *Jasse J. Garret* [06]. Označil ga je z besedami:

> *"Ajax je stil arhitekture na višjem nivoju načrtovanja, ki je sestavljena iz več med seboj povezanih tehnologij in idej"*.

Od tedaj je Ajax doživel velik razmah in hitro pridobival na priljubljenosti. S poslovnega, predvsem pa s tehnološkega vidika je izražal veliko neuresničljivega potenciala v tehnologijah spletnih brskalnikov. Podjetje *Google* in drugi glavni akterji so to izkoristili in pokazali, da je možno spletne aplikacije dvigniti na višjo raven delovanja. Njihovi uspešni produkti (*Google Mail, Google Maps, Flicker, itd.*) nam zagotovo služijo kot vzorčni primeri iz množice spletnih aplikacij, ki lahko svoj uspeh delno pripišejo prav tehnologiji Ajax.

V obdobju klasičnih spletnih aplikacij je nastal velik razkorak z namiznimi aplikacijami. Te so se hitreje razvijale in vse bolj napredovale. Uporabniki klasičnih spletnih aplikacij so bili prikrajšani za boljšo interaktivnost[1] in uporabniško izkušnjo, ki so jo omogočale namizne aplikacije. V zadnjih letih se delež klasičnih spletnih aplikacij vztrajno zmanjšuje in na njenih temeljih nastajajo vedno bolj sofisticirani spletni servisi. Njihov namen je zapolniti nastalo vrzel z uporabo obogatenih, bolj odzivnih, uporabniku prijaznejših ali kako drugače izboljšanih uporabniških vmesnikov, kar je ključnega pomena za odjemalce. S pristopom Ajax je to zagotovo mogoče, in to le s tehnologijami, ki so že nameščene in so del vsakega modernejšega spletnega brskalnika. V prvem delu naloge bomo pogledali razlike med obogatenimi spletnimi in namiznimi aplikacijami ter primerjali modela klasičnih in

[1] Interaktivnost (interakcija oziroma model interakcije) opisuje spremembe sistema kot posledico delovanja uporabnika in prav tako obratno (vzajemno delovanje). Tu je pomembna sprotna odzivnost in neprestana izmenjava podatkov in informacij računalniškega sistema z uporabnikom.

obogatenih spletnih aplikacij. Predstavili bomo nekatere tehnologije, ki so trenutno del razvoja obogatenih spletnih aplikacij, in opisali bomo njihove razlike.

Ko govorimo o napredku pri razvoju spletnih aplikacij na področju pristopa Ajax, ne smemo mimo dejstva, da imajo pri tem veliko vlogo vse vrste Ajax orodjarn, knjižnic in ostalih ogrodij. Z novejšimi in izboljšanimi različicami strmo narašča njihova priljubljenost med razvijalci. Zanje se odločajo manjše razvojne skupine in tudi večja podjetja. V zadnjih treh letih smo se srečali z nekontrolirano rastjo orodjarn Ajax. Zaradi tega je odločitev razvijalcev, katera orodjarna je primernejša za določen spletni projekt, precej težka. Pri tem je potrebno pretehtati dejstva, povezana z vprašanji: ali orodjarna podpira novejše brskalnike; ali obstaja skupina razvijalcev, ki skrbi za razvoj orodjarne in pogosto izdaja popravke ter nove različice; so funkcije dovolj dobro dokumentirane in je dokumentacija dovolj uporabna; se da hitro osvojiti potrebno znanje za delo z orodjarno; ima dovolj veliko in aktivno skupnost, v kateri sodelujejo njeni uporabniki in si s tem pomagajo; je mogoče orodjarno razširiti z dodatnimi funkcijami; je orodjarna dovolj dozorela za uporabo.

V primeru uporabe pristopa Ajax ali drugih tehnologij, ki se na trgu pojavljajo kot njegova alternativa pri izdelavi obogatenih spletnih aplikacij, je pomembno preučiti tudi vprašanje njihove produktivne uporabe. V ta namen bomo, poleg osnovne izvedbe
Ajax-a, preučili delovanje dveh orodjarn Javascript. S primerjavo njunih funkcionalnosti bodo izpostavljene razlike v ključnih konceptih, ki jih orodjarni implementirata, in tudi primernost njihove uporabe.

Najprej bomo spoznali pristop Ajax iz tehničnega vidika in z njim povezane pripadajoče tehnologije, ki zaokrožujejo njegov pristop v celoto. Namen tega je predvsem podati njihovo osnovno vedenje. To bo služilo kot pomoč pri razumevanju delovanja orodjarn Ajax in z njimi povezanih izzivov, s katerimi se razvijalci soočajo ob seznanjanju z njegovim pristopom.

2 TREND OBOGATENIH SPLETNIH APLIKACIJ

Obogatene spletne aplikacije postajajo v današnjem času vedno bolj priljubljene in nič ne kaže, da se ta trend ne bi nadaljeval tudi v prihodnosti. Prav tako je pristop Ajax - ena od tehnik za izdelavo obogatenih spletnih aplikacij - del tega trenda. Zaradi tega si bomo v tem poglavju v pogledali, kaj so obogatene aplikacije in katere tehnologije za njihovo izdelavo poznamo. Primerjali jih bomo z običajnimi spletnimi in namiznimi aplikacijami. Izpostavljene bodo nekatere njihove prednosti in slabosti.

2.1 Aplikacije *RIA* - nadgradnja spletnih in namiznih aplikacij

Obogatene spletne aplikacije so vmesni člen med namiznimi in spletnimi aplikacijami. V ta namen bomo naprej predstavili razlike med obema vrstama aplikacij.

Namizne aplikacije so, v primerjavi z običajnimi, bolj odzivne in interaktivne ter so zagotovo v prednosti, ko govorimo o grafičnih ter avdio in video vsebinah. Njihova uporaba ne zahteva nalaganja podatkov iz oddaljenega računalnika oziroma strežnika, saj je aplikacija lokalno nameščena. Zato so takojšnji odzivi na uporabnikove akcije. Slabost takih aplikacij se pokažejo v preveliki odvisnosti od sistema, na katerem so nameščene, visokih stroških vzdrževanja, licenc.

Tradicionalne spletne aplikacije temeljijo na jeziku *HTML* in imajo prednosti v lastnostih, v katerih imajo namizne aplikacije svoje šibkosti. So dostopnejše, ker omogočajo dostop preko medmrežja. Njihovo vzdrževanje je enostavnejše, ker so popravki in posodobitve na strežniku lahko nameščeni le enkrat in so takoj na voljo vsem uporabnikom. Take aplikacije niso odvisne od platforme, saj so zgrajene na osnovi spletnih standardov, ki omogočajo izvajanje znotraj spletnih brskalnikov. Zagotovo imajo tudi nekatere slabosti: niso tako odzivne in interaktivne, niso primerne za vizualizacijo in manipulacijo kompleksnejših podatkov. Precej moteče je tudi vsakokratno ponovno nalaganje vseh podatkov za vsak zahtevek.

Prva omemba izraza obogatenih spletnih aplikacij (v nadaljevanju naloge aplikacije *RIA*) je bila zabeležena v članku leta 2002, in sicer podjetja *Macromedia*, z naslovom: »*Naslednja generacija obogatenih odjemalcev*« [01]. Daje nam osnovni opis, kaj naj bi aplikacije *RIA* ponujale in katere pomanjkljivosti tradicionalnih spletnih aplikacij naj bi reševale. Aplikacije *RIA* poskušajo združiti prednosti namiznih in običajnih aplikacij ter izločiti njihove slabosti. Stabilnost, zrelost, razširljivost in usmerjenost v najnovejše tehnologije so vrline novega vala aplikacij *RIA*. Uvajajo bogatejšo uporabniško izkušnjo, naprednejše, bolj intuitivne in odzivne uporabniške vmesnike (npr. hitro se odzivajo na uporabnikove ukaze, prikazujejo trenuten položaj stanj, napredek opravljenih nalog...). Celoten model delovanja uporabnika z aplikacijo je postavljen na višjo raven. Prednosti, ki jih prinašajo aplikacije *RIA*, so:

- ***naprednejše uporabniške izkušnje.*** Te zagotavljajo obogaten uporabniški vmesnik, ki se prenese ob prvotnem uporabnikovem zahtevku. Ob vsakem naslednjem se prenese le ustrezen, posodobljen del, za razliko od običajnih spletnih aplikacijah, kjer se vsakokrat prenesejo vsi podatki. Tak uporabniški vmesnik mora zagotoviti

interaktivne elemente (kontrola povleci in spusti, drevesni pregled, razvrščajoči seznami...), če hočemo povečati podporo medsebojnega delovanja in narediti aplikacijo, bolj podobno namiznim aplikacijam;

- ***izboljšana interaktivnost sistema.*** V nasprotju z običajnimi spletnimi aplikacijami se tu lahko nekateri zahtevki v celoti obravnavajo na odjemalčevi strani, brez dodatne strežniške komunikacije. Ti se izvršijo takoj. Ostali, pri katerih je potrebna dodatna strežniška komunikacija, se izvršijo asinhrono tako, da to ne ustavi izvajanja celotne aplikacije (trenutno delo uporabnika se ne prekine). Odgovor na asinhron zahtevek vsebuje le del osveženih podatkov, s čimer je zagotovljena podobna zmogljivost in hitrost delovanja, kot jo poznamo pri namiznih aplikacijah;
- ***zmanjševanje omrežnega prometa.*** Večji del aplikacije se prenese le ob začetnem zagonu (ob njeni prvi uporabi). Ustrezno delno posodabljanje (in ne posodabljanje celotne aplikacije) se vrši glede na uporabnikove zahtevke. Zato je pri prvotnem nalaganju količina omrežnega prometa in čas samega nalaganja, v primerjavi z običajnimi spletnimi aplikacijami, lahko večja. Povečan promet in čas nalaganja sta posledica prenosa aplikacijske logike in kompleksnejših elementov uporabniškega vmesnika na odjemalčevo stran, vendar sta skupni čas nalaganja in količina omrežnega prometa še vedno manjša, kot sta pri običajnih spletnih aplikacijah;
- ***ni nameščanja aplikacije.*** Ker aplikacij *RIA* ni potrebno nameščati, je njihova uporaba prijaznejša uporabniku in njena razširjenost precej večja, kot je pri tradicionalnih namiznih aplikacijah;
- ***večja dostopnost***. Dostopnost je na višji ravni, kot je to pri običajnih spletnih aplikacijah, saj so te prav tako dosegljive preko medmrežja;
- ***nižji stroški vzdrževanja.*** Stroški vzdrževanja so prav tako nizki kot pri običajnih spletnih aplikacijah, saj ni potrebno lokalno nameščanje aplikacije. Zaradi tega tudi ni potrebno razdeljevati posodobitev ter skrbeti uporabniku za njihovo nameščanje.

Aplikacije *RIA* imajo določene pomanjkljivosti. Če hočemo zagotoviti, da bodo prednosti pri izvedbi določenega projekta v večji meri prevladale, jih moramo natančno preučiti. Pomanjkljivosti, s katerimi se srečujemo, so:

- ***daljši časi nalaganja***. Obogatene spletne aplikacije, v nasprotju z običajnimi, potrebujejo določeno mero odjemalčeve dodatne programske logike. Posledica tega je daljše prvotno nalaganje, trajanje tega je odvisno od količine potrebnih podatkov na začetku in tistih, ki so lahko na uporabnikovo zahtevo kasneje preneseni;
- ***posebna okolja za izvajanje.*** Logika aplikacije, ki se izvaja na odjemalčevi strani, lahko zahteva posebno okolje za izvajanje aplikacije (npr. Javanski virtualni mehanizem – v nadaljevanju *JVM*[2]). Lahko so to dodatni vtičniki za spletni brskalnik (npr. predvajalnik *Flash*) ali katere druge njegove posebne funkcionalnosti (npr. omogočen

[2] *JVM* – Javanski virtualni stroj ali mehanizem je potreben za izvajanje aplikacij, izdelanih s programskim jezikom Java.

mora biti jezik Javascript, podpora objektu *XMLHttpRquest*[3] - v nadaljevanju objekt *XHR*). Pomanjkanje tega lahko vodi v nedelovanje aplikacije na odjemalčevi strani ali povzroči nepričakovano obnašanje zaradi nepodprtih funkcij drugače implementiranega brskalnika;

- ***zahteva več sistemskih virov.*** Zaradi dejstva, da je na odjemalcu več programske logike in izvajanja procesiranja kot pri prikazovanju enostavne strani *HTML*, je potrebnih več sistemskih virov. Tako zasnovana arhitektura aplikacij *RIA* v večji meri obremenjuje odjemalčevo stran, kar posledično zmanjšuje obremenitve strežnika. Izvajanje teh, v kombinaciji še z drugimi, lahko povzroči težave pri napravah z manj procesorske moči;
- ***zmanjšana dostopnost.*** Aplikacije *RIA*, zaradi izvajanja znotraj brskalnika, navadno nimajo omogočenega dostopa do sistemskih virov. Na primer, aplikacija nima omogočenega dostopa do datotečnega sistema, v kolikor ji uporabnik ne zagotovi posebnih privilegijev. To lahko vodi v situacijo, ko uporabnik noče zagotoviti privilegijev iz varnostnih razlogov, brez katerih aplikacija ne more pravilno delovati;
- ***vsebina ni vidna iskalnikom.*** Velika prednost običajnih spletnih aplikacij je, da jih spletni iskalnik lahko najde. To je doseženo z indeksiranjem glavne strani in njenih podstrani. Ker so vsebine aplikacij *RIA* predstavljene kot ena spletna stran, običajno niso dostopne preko enoličnih naslovov *URL*[4]. To onemogoča pravilno izvedbo indeksiranja, zaradi česar je njihova vsebina velikokrat nevidna spletnim iskalnikom. Ta problem lahko zmanjšamo z učinkovitejšim optimiziranjem spletne strani za iskalnike. Aplikacije *RIA* postajajo vedno bolj priljubljene in v prihodnosti je odgovornost tudi na iskalnikih, da upoštevajo ta trend in prilagodijo svoje iskalne strategije tudi njim;
- ***problem z uporabo zgodovine.*** Naslednja stvar, ki pogosto zmede uporabnika, je mehanizem delovanja zgodovine v brskalniku. Ker so aplikacije *RIA* predstavljene kot ena spletna stran, njihov spletni naslov ne določa enolično točnega stanja aplikacije. Zaznamovanje take spletne strani je tako onemogočeno, saj je malo verjetno, da bomo ob naslednji uporabi tega zaznamka pridobili aplikacijo v enakem stanju. Nepravilno je delovanje gumba »nazaj« v spletnem brskalniku. Njegova uporaba lahko povzroči prikaz spletne strani, ki je bila zahtevana pred začetkom izvajanja trenutne aplikacije in ne njenega prejšnjega stanja. To velikokrat ni uporabnikov namen. Nekatere tehnologije ponujajo lastne rešitve tega problema, vendar so te redko uporabljene. Rešitev, kako zaobiti tak problem, je v večini primerov odvisna od razvijalce aplikacij samih oziroma jo je možno rešiti z uporabo orodjarn Ajax.

---

[3] Objekt *XMLHttpRequest* je podrobneje opisan v razdelku 3.3.3.

[4] Je enoličen naslov, ki se običajno začne s *http://www* in z njim povemo brskalniku, kje naj poišče spletni dokument.

Tabela 2.1 povzema primerjavo med namiznimi in običajnimi spletnimi aplikacijami. Ta prikazuje, v katerih lastnostih imata oba tipa aplikacij prevladujočo moč in v katerih šibkost ter kako poskušajo združiti prednosti obeh obogatene spletne aplikacije.

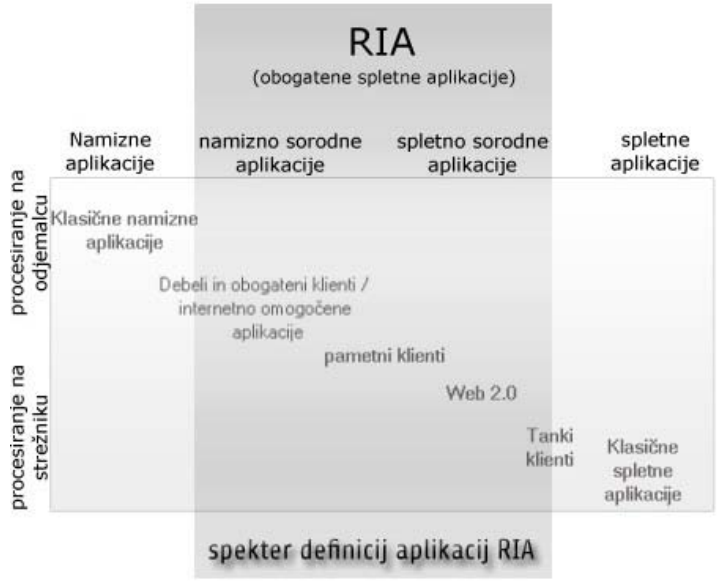
Lastnost	Namizne aplikacije	Spletne aplikacije	Obogatene spletne aplikacije
Bogatejša uporabniška izkušnja	+	-	+
Interaktivnost, odzivnost	+	-	+
Manjši obseg vzdrževanja	-	+	+
Visok doseg	-	+	+

Tabela 2.1: Značilnosti namiznih in spletnih aplikacij, združenih v aplikacijah *RIA* [02].

Osnovni izziv, s katerim se soočimo pri razvoju obogatenih spletnih aplikacij, je, da v največji možni meri izkoristimo prednosti in zmanjšamo slabosti. Za večino so na voljo rešitve, vendar so povezane z vloženim trudom. Zato je uspeh posamezne aplikacije odvisen tudi od meril, ki so ključnega pomena, predstavljenih v tem razdelku.

2.2 Delitev obogatenih spletnih aplikacij

Za izdelavo aplikacij *RIA* imamo na voljo precejšnje število različnih tehnologij. Njihovo razlikovanje je mogoče zaznati predvsem po količini opravljenega procesiranja na strežnikovi strani oziroma na strani odjemalca. Uporaba terminologije za namizne, spletne ter vse ostale vmesne tehnologije se včasih zdi težavna. Njihovo razvrščanje v posamezne skupine ni tako nedvoumno, kot bi si mislili na prvi pogled. Uporabljena terminologija zajema izraze, kot so: namizne aplikacije, obogateni ali debeli klienti, internetno-omogočene aplikacije, pametni klienti, tanki klienti, spletne aplikacije, spletne strani. Za velik del teh se uporablja kar izraz aplikacije *RIA*.

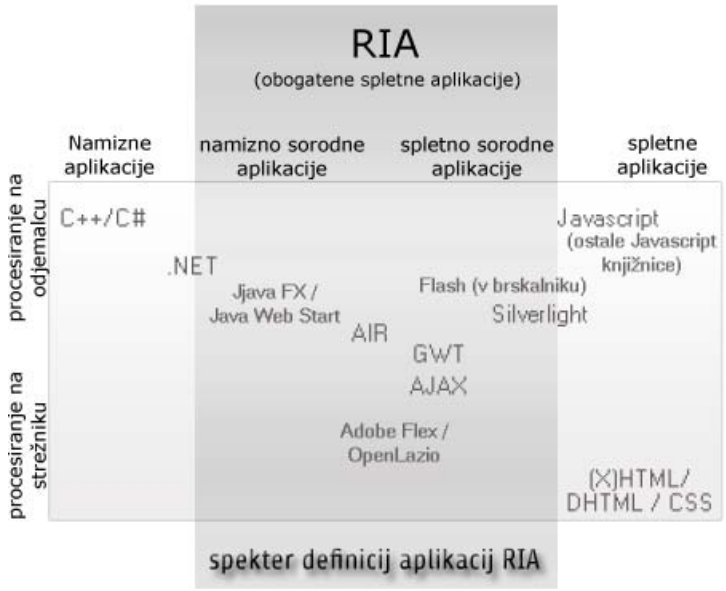


Slika 1: Kategorizacija aplikacij *RIA* glede na mesto procesiranja

Natančnejša delitev definicije aplikacij *RIA* zajema širok spekter različnih tipov aplikacij (Slika 1). Krajši opis nam bo omogočil razumeti njihovo osnovno idejo. Razvrščamo jih v naslednje kategorije:

- ***namizne aplikacije***. So aplikacije, ki jih je potrebno pred prvotno uporabo namestiti v operacijskem sistemu. Klasične namizne aplikacije so, na primer, aplikacije za obdelavo besedil (npr. *Microsoft Word*), za grafično obdelavo (npr. *Adobe Photoshop*) in številne druge;
- ***internetno-omogočene aplikacije.*** Aplikacije s povezavo v omrežje imenujemo internetno-omogočene aplikacije. Te delujejo s podporo medmrežja, kar pa ni pogoj za njihovo delovanje, lahko delujejo brez nje. Še vedno so nameščene lokalno in se zaradi tega lahko izvajajo, vendar je njihova funkcionalnost omejena. Primer takih aplikacij so odjemalci elektronske pošte, npr. *Mozilla Thunderbird ali Microsoft Outlook;*
- ***obogateni klienti.*** Z njimi lahko razvijalci izdelajo lastne aplikacije na obstoječi platformi. Namesto izdelave aplikacije od popolnega začetka se izkoristijo preizkušene funkcionalnosti ogrodij, ki jih zagotavlja platforma [16]. Primer take aplikacije je, npr. orodje *NetBeans*, okolje za enoten razvoj aplikacij;
- ***spletne strani, spletne aplikacije, tanki klienti.*** Na drugi strani imamo spletne strani oziroma aplikacije in tanke kliente. Te vrste aplikacij navadno ne zahtevajo nameščanja z uporabnikove strani. Delujejo preko omrežja in se ponavadi zaženejo iz spletnega brskalnika. Zaradi tega je obvezna povezanost v medmrežje, brez katerega je onemogočeno njihovo delovanje [19];
- ***pametni klienti.*** Ta izraz se je pojavil v zadnjem času in je močno povezan z aplikacijami *RIA*. Je zvrst tehnologije med tankimi in debelimi klienti.

Razberemo lahko, da so aplikacije, ki so bolj *RIA*-usmerjene, nahajajo v navpičnem pravokotniku, to je v sredini med namizno in spletno sorodnimi aplikacijami, ostale so zunaj njega (Slika 2). Na navpični osi jih razvrščamo glede na količino opravljenega procesiranja na strani strežnika in odjemalca. Vodoravna os služi razvrščanju posameznih tipov aplikacij glede na večjo usmerjenost k namiznim oziroma spletnim aplikacijam. Na desno stran so razvrščene tiste aplikacije, ki so bolj spletno usmerjene in imajo vse značilnosti aplikacij *RIA*. Omogočajo, na primer, uporabnikovo personifikacijo, kot so razna socialna omrežja (npr. *Facebook)*, spletni koledarji (npr. *Google Calendar*), spletni servisi za elektronsko pošto (npr. *Yahoo Mail, Google Gmail, itd.*). Na drugi strani imamo aplikacije *RIA*, ki po izgledu in občutku (zaradi uporabniškega vmesnika) delujejo kot namizne aplikacije. V skupino namizno-usmerjenih aplikacij z velikim poudarkom na spletnem delovanju sodi npr. predvajalnik video in avdio vsebin *Apple iTunes*. Nameščen mora biti lokalno, zato daje vtis namizne aplikacije. Z drugega vidika ga lahko označimo za spletno aplikacijo, saj omogoča spletno nakupovanje. V tem kontekstu lahko to in druge, tej podobne, aplikacije označimo tudi s poimenovanjem obogateni klienti, ki se uveljavlja v zadnjem času. Vidimo, da so meje med temi tehnologijami zamegljene in njihovo jasno razvrščanje ni eksaktno, kar pomeni, da se lahko razlikuje glede na lastno interpretacijo.



Slika 2: Širši spekter definicije aplikacij *RIA*, določen z različnimi tehnologijami

V okviru definicije aplikacij *RIA* poznamo še delitev, ki se je uveljavila v današnjem času. Temelji na platformah za njihovo izvajanje. Pretežni del aplikacij *RIA* temelji na platformi spletnega brskalnika, vendar so se uveljavili tudi drugi načini [20]. Ti so:

- ***delovanje znotraj brskalnika. To*** je pristop, tako imenovanega, »čistega« brskalnika, poznanega kot pristop Ajax. Vključuje samo tiste tehnologije, ki so vsebovane že v vsakem modernejšem brskalniku. Ta način podpirajo številne orodjarne Javascript;
- ***razširjeno delovanje brskalnika. To*** je pristop, ki potrebuje za svoje delovanje, poleg brskalnika, ustrezno razširitev z vtičniki ali drugimi, brskalniku specifičnimi, dodatki.

Kot primere takega delovanja bi lahko omenili *Adobe Flash* (z dodatno strežniško tehnologijo *Adobe Flex*), *Microsoft Silverlight* …;

– ***delovanje izven brskalnika.*** Je pristop, ki zahteva namestitev določene programske opreme za vzpostavitev delovnega okolja na odjemalčevi strani. Primera takih okolij sta npr. *JVM*, *AIR* (Adobovo integrirano okolje za izvajanje). To zagotavlja podlago za delovanje aplikacij, kot sta *Adobe AIR ali JavaFX*.

V nadaljevanju bomo predstavili tehnologije omenjenih kategorij. Ne bomo se spuščali v njihovo podrobno delovanje, saj to presega okvire diplomske naloge. Večji poudarek bomo dali le pristopu Ajax, ki nam bo kasneje služil kot osnova za razumevanje delovanja orodjarn Javascript, ki temeljijo prav na njem, saj zajema tehnologije za izdelavo, tako imenovanih, spletnih aplikacij »*Web 2.0*«. Vidimo lahko, da uporablja ta tip aplikacij tehnologije, ki so bolj spletno sorodne (delujejo znotraj brskalnika, uporabljajo tehniko asinhronega osveževanja, uporabljajo orodjarne Javascript) (Slika 2).

Ostale tehnologije aplikacij *RIA* so se v večini razvijale vzporedno s pristopom Ajax oziroma je v zadnjem obdobju večji poudarek na njihovem razvoju. Njihov kratek opis bo zadoščal, da se seznanimo z njimi in vemo, da obstajajo in jih lahko uporabimo kot možno alternativo pristopu Ajax.

Ni potrebno omejevati se pretežno le na eno platformo, ampak je bolje, da se teži tudi k razvoju drugih. Pokazatelji tega so že nekatere nove spletne aplikacije, ki se uveljavljajo na trgu aplikacij *RIA*. Njihovo izvajanje ne temelji več na spletnem brskalniku, ampak na posebej nameščenem okolju za izvajanje. Zdi se, da lahko spletni brskalnik, ki je danes glavna platforma za izvajanje aplikacij *RIA*, v prihodnosti povsem izgine oziroma počasi izgubi prevladujočo vlogo.

Nov način interakcije in drugačni uporabniški vmesniki ter ostale novosti, ki jih uvajajo aplikacije *RIA*, bodo v začetku zahtevali od uporabnika določen čas prilagajanja. To verjetno ne bo všeč prav vsem. Vključevanje novih komponent bo moralo biti premišljeno od razvijalcev aplikacij, če bodo hoteli ohraniti, kar se da preproste aplikacije. Z njihovo pretirano uporabo bi lahko preveč obremenili delovanje in zmanjšali učinkovitost aplikacij. S tem bi dosegli nasproten učinek in večino uporabnikov odvrnili od njihove uporabe. Prav zato je pomembno ustvarjati uporabniku prijazne aplikacije, ki odpravljajo nezadovoljstvo uporabnikov in zmanjšujejo omenjena tveganja.

2.1 Tehnologije za razvoj obogatenih spletnih aplikacij

Splošen pregled platform in tehnologij za razvoj aplikacij *RIA* zajema štiri bolj poznane platforme. Pri tem je brskalnik, skupaj s pristopom Ajax, obravnavan kot lastna platforma, ostale se pojavljajo kot njegova možna alternativa. Pregled temelji na podlagi primerjave navedenih meril, s pomočjo katerih so izpostavljene prednosti in slabosti primerjanih platform:

– ***uporabniška izkušnja.*** Stopnja interaktivnosti in odzivnost, ki jo ponuja posamezna platforma;

- ***neodvisnost od platforme***. Izvajanje aplikacij ni omejeno samo na določene operacijske sisteme, spletne brskalnike, okolja za izvajanje …, ampak ni odvisno od tega;
- ***postavitev.*** Potrebno je nameščanje, nadgrajevanje aplikacij oziroma kakšne druge značilnosti za njihovo delovanje;
- ***mešanica tehnologij.*** Je uporabljena ena sama tehnologija ali je potrebna mešanica več tehnologij;
- ***zagon.*** Potreben čas, preden lahko uporabnik začne uporabljati aplikacijo, tako imenovan, čas inicializacije (vzpostavitve okolja). Ta je tudi odvisen od hitrosti omrežne povezave uporabnika;
- ***odjemalec / strežnik.*** Količina procesiranja na strani odjemalca in na strani strežnika;
- ***obogaten uporabniški vmesnik.*** Oblikovanje delovanja uporabniškega vmesnika in njegovih komponent;
- ***razsežnost.*** Razširjenost med uporabniki***;***
- ***podpora razvijalcu.*** Podpora pri razvoju aplikacij (različne vrste orodji, razvojne tehnike, seznanitev/sprejemanje tehnologij razvijalcev, itd.;
- ***dostopnost*** (spletnim brskalnikom). Dostopnost samodejnega indeksiranja spletnega iskalnika, zmožnost dostopa ljudi s posebnimi potrebami;
- ***stroški.*** Stroški razvoja, licenciranja, odprto-kodne oziroma lastniške rešitve;
- ***posebne značilnosti.*** Obstoj kakšnih posebnih značilnosti.

2.1.1 Brskalnik kot platforma

Pri brskalniku kot platformi bi lahko upoštevali različne načine razvoja aplikacij, od uporabe čisto navadnega jezika *(x)XHTML*, tehnik dinamičnega *HTML-ja* in nenazadnje, uporaba tehnike Ajax. Ta je najnovejša, zato je primerna za primerjavo.

Tabela 2.2: Prednosti in slabosti brskalnika kot platforme [17, 07].

	+	-
Uporabniška izkušnja	Zdi se revolucionarna, če jo primerjamo s tehniko DHTML	Shranjevanje zgodovine dinamično prikazanih strani; slabše podprta navigacija v brskalniku; spletnih naslovov ne določajo enolično trenutno prikazanega stanja aplikacije; manjša uporabnost uporabniških vmesnikov kot pri konkurenčnih tehnologijah
Neodvisnost od platforme	Široko podprt na različnih okoljih	različna interpretacija Javascript in CSS-kode med različnimi brskalniki; izvajanje v »eksotičnih« brskalnikih večinoma ni preizkušeno
Postavitev	Osrednja strežniška postavitev; ni potrebe po nameščanju ali posodabljanju na odjemalčevi strani; manjša ovira za uporabnika, da pride v stik z	Na strani odjemalca je lahko Javascript izključen v brskalniku; izvorno kodo v jeziku Javascript je težko skriti

	njim	
Mešanica tehnologij	Zmogljiva kombinacija tehnologij	Ni zagotovljena popolna standardizacija in interpretacija tehnologij med različnimi brskalniki.
Zagon	Relativno hiter glede na vključenost tehnologij v brskalnik	Včasih je potrebno naložiti večjo količino izvorne kode v Javascript, kar lahko upočasni zagon.
Odjemalec / strežnik	V primerjavi s tehnologijo *DHTML,* gre tu za asinhrono komunikacijo; znižanje strežniške obremenitve – to je pošiljanje le pravkar nastalih delnih zahtevkov	Ne ponuja povezovanja s pomočjo vtičnic; zahteva nekaj več procesorske moči od odjemalca.
Obogaten uporabniški vmesnik	Ločeno oblikovanje od logike aplikacije z uporabo stilov; model interakcije, ki ga ponuja uporabniški vmesnik, je precej boljši od tistega, ki je pri tradicionalnih spletnih aplikacijah	Težko je izvesti nekatere običajne značilnosti, ki so del naprednejšega uporabniškega vmesnika (zavihki, drevesni pregled, tehnika povleci in spusti, drsniki, itd.); pomanjkanje enotnega koncepta *»videz in občutek«*[5]
	Možna izboljšava z uporabo Javascript orodjarn	
Razsežnost	Vključena v brskalniku in tako dokaj razširjena; ni potrebe po dodatnih vtičnikih; izvedljiv skoraj na vseh platformah	
	Vsi novejši brskalniki podpirajo Ajax (približno od leta 2000)	
Podpora razvijalcu	Veliko prosto dostopnih orodji (npr. Firebug, urejevalniki programske kode); pri večjih projektih je potreba po uporabi integriranega razvojnega okolja; možna uporaba skupnosti kot pomoč pri delu s posameznimi tehnologijami in reševanju težav pri razvoju	Oteženo zagotavljanje enakega izvajanja izvorne kode posameznih tehnologij v različnih spletnih brskalnikih.
Dostopnost (spletnim iskalnikom)	Dobra struktura *HTML-ja* je pomembna za doseganje višjih rangov v iskalnikih; boljše indeksiranja spletnih iskalnikov	Vsebina *DHTML* bo prezrta od spletnih iskalnikov; prav tako bo z dinamično naloženo vsebino.
Stroški	Tehnologije in orodja so prosto dostopna; manjši zaradi hitrejšega in lažjega razvoja	Večji, ker zagotavlja različne implementacije aplikacije zaradi razlik v delovanju brskalnikov (ne povsem standardizirane tehnologije).
Posebne značilnosti	Dober pristop pri razvoju *RIA* zaradi tehnologij, že vključenih v brskalnik; manjša ovira za uporabnika, da pride v stik s tako aplikacijo	Za izvedbo obsežnejših projektov potreba po uporabi strežniških tehnologij (*PHP, Python).*

[5] (anlg. Look and Feel) - se uporablja za grafični uporabniški vmesnik in zajema vidike njegove zasnove, vključno z elementi, kot so: barve, oblike, postavitve in pisave (videz) ter obnašanje dinamičnih elementov, kot so: gumbi, meniji (občutek)

2.1.2 Flash kot platforma

Razvoj, ki temelji na platformi *Flash,* vsebuje skupino tehnologij (npr. *Adobe Flex, AIR*) podjetja *Adobe Systems*. Te uvajajo aplikacije *RIA* s pomočjo dveh ključnih elementov, integriranega razvojnega okolja in razvojnih orodij.

Aplikacijah *RIA* zahtevajo stalno komunikacijo med odjemalcem in strežnikom ter obravnavo teh podatkov. To ni bil prvotni namen aplikacij *Flash*. Tako so se uporabljale predvsem za manjše animacije, oglaševanja. Delovale so na enostaven način, in sicer s prenosom ene ali več *Flash* datotek na stran odjemalca, kjer so se procesirale. Pri tem komunikacija s strežnikom ni bila več potrebna. V zadnjih letih se je podpora aplikacij *RIA* bolj uveljavila in razširila. Tako so se pojavile rešitve na strežnikovi strani, s katerimi so aplikacije *Flash* sposobne komunicirati in se pojavljajo tudi kot alternative tehnologiji Ajax.

Adobe Flex je ena izmed rešitev, s pomočjo katere lahko izdelujemo aplikacije *RIA*. Razvoj poteka v jeziku *ActionScript 3* (s koreninami v jeziku Javascript), ki ga lahko uporabljamo tudi brez avtorskih orodij. Tehnologija *Flash* je primerna za aplikacije, ki se izvajajo znotraj spletnega brskalnika. Možno je tudi samostojno izvajanje znotraj operacijskega sistema. Za to je na odjemalčevi strani potreben predvajalnik, znotraj katerega se izvaja aplikacija *Flash*.

Adobe AIR je novejša rešitev, ki se uporablja zunaj okolja brskalnika. Ponuja nam zagon aplikacije s pomočjo spletne povezave, ki ji sledi proces namestitve in integracije v namizno okolje. Ta tehnologija temelji na ogrodju *Adobe Flex*.

Tabela 2.3: Prednosti in slabosti tehnologije Flash kot platforme [17, 07, 22]

	+	-
Uporabniška izkušnja	Brskalnikov vtičnik *Flash* nudi boljšo uporabniško izkušnjo kot Ajax; *AIR* , integrirana namizna rešitev, omogoča izdatno (so)delovanje z namizjem.	
Neodvisnost od platforme	Okolje za izvajanje (*Flash* predvajalnik) je na voljo na večini platform.	
Postavitev	Osrednja strežniška postavitev; več rešitev na strani strežnika in s tem povezanih zmožnosti.	Samodejna nadgradnja novejših različic, ki pa lahko večkrat prekine uporabnikov delovni tok.
Mešanica tehnologij	Enotna tehnologija zagotavlja enako delovanje na vseh platformah.	Omejeno sodelovanje z drugimi tehnologijami.
Zagon	Glede na vtičnik je zagon relativno hiter; Možno je postopno nalaganje vsebin.	Potreben je zagon okolja za izvajanje aplikacij.
Odjemalec / strežnik	Zmožnost takojšnje interakcije z uporabnikom s pomočjo procesiranja na strani odjemalca; *Flex/AIR* tehnologijami se zmanjšuje obremenitev strežnika, ni odvečnih podatkov, ki bi se prenašali.	

	Omogočanje povezave z vtičnicami;	
Obogaten uporabniški vmesnik	Ogromne zmožnosti pri oblikovanju, animaciji, uporabi standardnih komponent (z uporabo tehnologije *Adobe Flex*); dobro uveljavljen glede na to.	Nezmožnost uporabe stilov za komponente, lastne operacijskemu sistemu.
Razsežnost	Eden najbolj razširjenih vtičnikov za spletne brskalnike	
Podpora razvijalcu	Uveljavljena orodja s standardnim *IDE-jem* in podporo z vtičniki (npr. orodje *Eclipse*).	Za razvijalce manj uporaben v splošne namene zaradi svoje usmerjenosti v grafično bogatejše aplikacije kot tiste z več logike.
Dostopnost (spletnim iskalnikom)		Vsebina ni vidna iskalnikom; ni ločenih spletnih naslovov za različna stanja v aplikaciji; težko dostopno invalidnim osebam.
Stroški	Za uporabnika brezplačen; na voljo so nekatere nadomestne, cenejše rešitve.	Orodja za razvoj so draga (tiste, ki zagotavlja podjetje *Adobe Systems*); zanimiva alternativa tehnologiji Adobe *Flex* je *OpenLaszlo*, ki ima prednost v tem, da je odprto-kodna platforma, brezplačna.
Posebne značilnosti	Odlična podpora avdio in video vsebinam.	

2.1.3 Microsoftova platforma – Silverlight

Silvelight je Microsoftova tehnologija za izdelavo obogatenih spletnih aplikacij, ki temelji na *.NET* ogrodju. Izhaja iz iste osnove kot klasične namizne aplikacije, ki so plod razvoja Microsoftovih tehnologij. Tehnologija *Silverlight* deluje v spletnih brskalnikih in ni namenjena uporabi za klasične namizne aplikacije, kot bi lahko pričakovali glede na njen izvor. Logika aplikacij se izvaja v jezikih *.NET* (*VisualBasic*, *C#*, itd). Za izdelavo uporabniškega vmesnika se uporablja označevalni jezik *XAML* (izhaja iz jezika *XML*), ki podpira uporabo stilov in predlog.

Tabela 2.4: Prednosti in slabosti tehnologije *Silverlight* kot platforme [11, 13]

	+	-
Uporabniška izkušnja	Kot brskalnikov vtičnik nudi *Silverlight* podobno uporabniško izkušnjo kot tehnologija *Adobe Flash.*	Ni podprt v nekaterih razširjenih brskalnikih (npr. *Opera*) – omejen na določene brskalnike.
Neodvisnost od platforme		Podprta sta le brskalnika *IE* in *Firefox* na sistemih Windows ter *Firefox* in *Safari* na *Mac* sistemih.
Postavitev	Osrednja strežniška postavitev; enostavno nalaganje ustvarjenih datotek na strežnik; potrebne so Microsoft strežniške rešitve.	Na voljo le za nekatere platforme.
Mešanica	Mogoča izvedba v več programskih jezikih.	Tehnologija, preveč odvisna od drugih.

tehnologij		Microsoftovih rešitev; včasih ni v skladu z mednarodno standardizacijo.
Zagon	Glede na vtičnik je zagon relativno hiter; možno postopno nalaganje vsebin.	Potreben je najprej zagon okolja za izvajanje aplikacij;
Odjemalec / strežnik	Zmožnost takojšnje interakcije (posredovanja) uporabniku s pomočjo procesiranja na strani odjemalca.	Primarno osredotočen na popolno izvajanje od odjemalca.
Obogaten uporabniški vmesnik	Zmožnost oblikovanja, različne animacije; uporaba standardnih komponent.	
Razsežnost	Majhna velikost (~ *2MB*) namestitvenega okolja za izvajanje *Silverlight* aplikacij in tako dokaj enostavna za nameščanje.	Lahko je pred nameščen samo na operacijskih sistemih *Microsoft Windows* ali spletnem brskalniku *Internet Explorer*.
Podpora razvijalcu	Enostaven za kvalificirane razvijalce, ki obvladajo *NET*; izbira med več programskimi jeziki; uveljavljena orodja.	Manj podpore za prosto dostopna in odprto-kodna razvojna okolja.
Dostopnost (spletnim iskalnikom)	Predstavitvena plast bi lahko bila dostopna spletnim iskalnikom, vendar ga trenutno ne upošteva.	Vsebina je skrita *(privzeto).*
Stroški	Brezplačen za uporabnika.	Precej draga orodja za razvoj, npr. *Visual Studio 2008.*
Posebne značilnosti	Kvalitetna podpora video in avdio vsebinam; omogoča grafično (*Direct3D*) pospeševanje s pomočjo strojne opreme.	

2.1.4 Java kot platforma

Tehnologija Java sestoji iz programskega jezika in okolja za izvajanje aplikacij. Pri razvoju aplikacij *RIA* jo lahko uporabljamo na več načinov. Novejši pristop, uporabljen v ta namen, je tehnologija *JavaFX*. V ozadju uporablja tudi jezik Javascript. Poslužuje se ga z razlogom podpore in izboljšave razvoja uporabniškega vmesnika. Namen tehnologije *JavaFx* ni vsesplošna uporabnost, kot je to znano za programski jezik Java, ampak se osredotoča bolj na interaktivnejšo in medijsko bogatejšo vsebino ter grafični uporabniški vmesnik. Pomembno je vedeti, da tehnologija *JavaFx* večinoma le združuje že obstoječe tehnologije Java. To se vidi v sodobnejšem in doslednejšem uporabniškem vmesniku. Izvajamo jo lahko na vsaki platformi, ki ponuja *JVM*.

Tabela 2.5: Prednosti in slabosti Jave kot platforme [17, 07]

	+	-
Uporabniška izkušnja	Uporabna znotraj in zunaj spletnega brskalnika; aplikacija je shranjena v pomnilnik – prenese se samo enkrat.	Daljši začetni prenos aplikacije; začetni zagon Jave.

Neodvisnost od platforme	Popolnoma neodvisna; okolje za izvajanje aplikacij je na voljo za glavne platforme.	
Postavitev	Osrednja strežniška postavitev.	
Mešanica tehnologij	Vse, kar ima, temelji na tehnologiji Java; dobra povezljivost; eno okolje za izvajanje aplikacij za vse platforme.	
Zagon	Možen zagon iz namizja.	Daljši proces zaganjanja aplikacij, sestavljenih iz več korakov.
Odjemalec / strežnik	Takojšnji odziv s pomočjo procesiranja na strani odjemalca; možna uporaba brez medmrežnega dostopa; povezovanje s pomočjo vtičnic; ni nepotrebnega nalaganja podatkov;	Okolje za izvajanje aplikacij Java potrebuje dodatno procesorsko moč na strani odjemalca; težavam se izognemo ob povečanju procesorske moči.
Obogaten uporabniškega vmesnika	Osnovan na komponentah, ki so podobne tistim v namiznih aplikacijah; podpora koncepta *»videz in občutek«* za standardne komponente*;* delo s stili bi lahko bilo lažje z uporabo boljših orodij.	
Razsežnost	Okolje za izvajanje aplikacije je precej razširjeno in uveljavljeno.	
Podpora razvijalcu	Mogoč splošno namenski jezik; prosto-dostopna in uveljavljena orodja za razvoj, ki podpirajo večino sodobnih pristopov (npr. *NetBeans* – podpora objektno usmerjenega razvoja s pregledovalnikom objektov, hierarhijo razredov, razhroščevalnikom kode, prevajalnikom kode).	
Dostopnost (spletnim iskalnikom)		Vsebina nevidna spletnim iskalnikom.
Stroški	Odprto-kodna; ni stroškov licenc.	
Posebne značilnosti	Odlično izpolnjuje poslovne potrebe; tehnologija *JavaFX* je tudi koristna za zmanjševanje stroškov pri integraciji med več napravami.	

Ta primerjava prikazuje razlike med različnimi platformami glede na merila, ki smo jih določili. Nekatera med njimi so pomembnejša od drugih. Ta se razlikujejo v odvisnosti od projekta, ki ga želimo izvesti. Zaradi tega je preučitev posameznih meril pomembna za izbiro ustrezne platforme. Lahko zaključimo, da je pristop Ajax (brskalnik kot platforma) še vedno ena bolj razširjenih platform na tržišču za razvoj aplikacij *RIA*. Kljub temu obstaja možnost,

da v prihodnosti izgubi primat, če ne bo sledila razvoju ostalih (alternativnih) tehnologij in znala odgovoriti na vsa pričakovanja, ki se pojavljajo v zvezi z razvojem aplikacij *RIA*.

2.2 Arhitektura obogatenih spletnih aplikacij

Arhitektura aplikacij RIA je v osnovi zgrajena po principu modela *MVC*[6] [16]. To je standardna arhitektura, široko uporabljena pri izdelavi vseh vrst interaktivnih aplikacij. Pri razvoju spletnih aplikacij odjemalec/strežnik so komponente tega modela razporejene na vsaj dve mesti v omrežju. Tako se del aplikacijske logike izvaja na odjemalčevi strani (del aplikacije, ki je v interakciji z uporabnikom), ostala aplikacijska logika pa se izvaja na strežnikovi strani (del, kjer poteka interakcija s podatki). Pojavlja se vprašanje, kako razporediti posamezne komponente tako zasnovane arhitekture in kako upravljati z njihovo komunikacijo. Prva rešitev temelji na modelu *MVC*, ki je zasnovan le na strežniški osnovi in se uporablja pri tradicionalnih spletnih aplikacijah. Ta rešitev je že uveljavljena. Vendar se v tem primeru spletni brskalnik obnaša kot pasivni terminal. Vse informacije se držijo na strežniški strani, tipično v uporabnikovi seji. Ko se uporabnik prijavi ali kako drugače inicializira sejo, se ustvarijo določeni objekti, ki predstavljajo npr. nakupovalno košarico, priporočene artikle, če je to stran spletne trgovine. V istem trenutku se servira spletna stran v uporabnikovem brskalniku in se ob vsakokratni njegovi novi interakciji ponovno ustvari. Spletni brskalnik zavrže staro stran, saj kot pasivni terminal z njo nima kaj početi.

Druga rešitev, uporablja se v aplikacijah *RIA*, je razširjena različica prvotne rešitve. Razlikuje se v izvajanju modela *MVC*, katerega del aplikacijske logike je vključeno tudi na odjemalčevi strani. Na primer, pri prijavi uporabnika je prenesen kompleksnejši spletni dokument, katerega večji delež zavzema programska logika Javascript. Ta dokument ostane z uporabnikom celoten čas trajanja seje, čeprav se lahko znatno spremeni ob stiku uporabnika z njim. Ker se dokument ohrani čez celotno uporabnikovo sejo, je možno hraniti stanja v spletnem brskalniku in ne več v seji na strežnikovi strani.

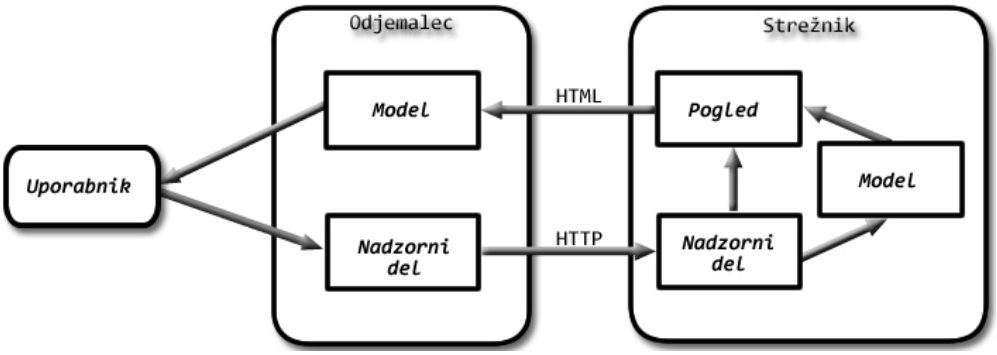
Pri tej rešitvi je potrebno sprejeti določene arhitekturne odločitve glede razporeditve posameznih komponent, ki imajo tudi nekatere skupne značilnosti z rešitvijo, že prej omenjeno, zaradi naslednjih temeljnih pravil:

- vsaj del arhitekture MVC (komponenta modela) se mora izvajati na strežnikovi strani. Ta je odgovoren za vzdrževanje integritete podatkovnih objektov in mora biti imun na razne prevare na odjemalčevi strani;
- uporabnik mora videti podatke, predstavljene s strani komponente pregleda. To pomeni, da je njegova naloga, ne glede na mesto izvajanja, kontroliranje prikaza informacij na strani odjemalca;

[6] (angl. Model, View, Controler) – opisuje ločitev delov programske kode v tri različne dele. Vloga posameznih delov je sledeča: model predstavlja domeno problema aplikacije in programska koda se tu nanaša predvsem na delo s podatki in podatkovno bazo; pogled služi za predstavitev uporabniškega dela uporabniku; nadzorni del služi nadziranju operacij, ki potekajo med predhodno omenjenima komponentama, saj te naj ne bi komunicirale direktno med seboj, temveč vedno preko nadzornega dela.

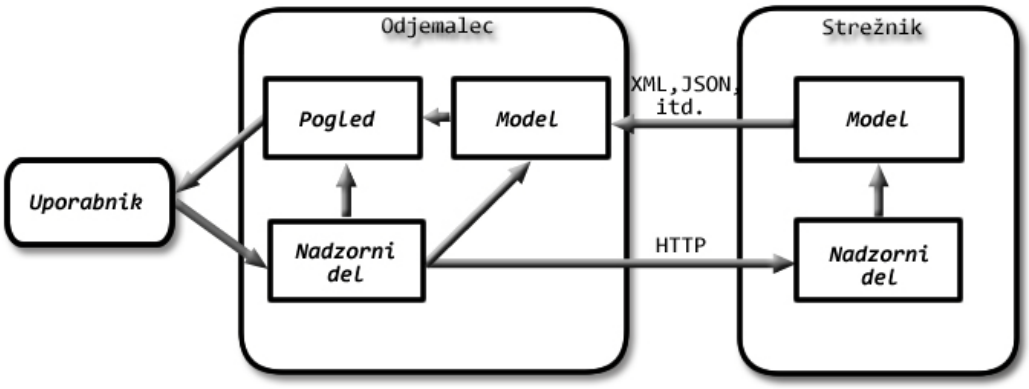
– nadzorna komponenta mora biti sposobna prejemati dogodke, ki predstavljajo uporabnikovo delovanje v aplikaciji.

Pri modelu *MVC*, ki temelji samo na strežniški osnovi, se vse njegove komponente (pogled, model, nadzorni del) izvajajo na strežniku (Slika 3). Pogled ustvari predstavitev podatkov (spletno stran). Ta je prenesena kot odgovor na odjemalčevi strani, kjer je prikazana. Delovanje uporabnika se v spletnem brskalniku zazna preko spletnega naslova s pomočjo parametrov, ki so poslani na strežnik v zahtevi *HTTP*. Nadzorni del na strežniku interpretira dobljene podatke in posreduje zahtevo komponenti pogleda in modela. Rezultat tega je novo ustvarjena predstavitev (spletna stran), ki je v celoti prenesena na odjemalčevo stran. Pri običajnih spletnih aplikacijah je posodabljanje komponente pogleda izvedena kot celotna zamenjava prejšnjega pogleda z novejšim. To zaznamo v spletnem brskalniku kot osvežitev spletne strani.



Slika 3: Prikaz arhitekture *MVC* tradicionalnih spletnih aplikacij

Spletni brskalnik je združen v monolitno celoto z odjemalcem (Slika 3, 4). Prikazana arhitektura modela *MVC* torej skriva podrobnost interakcije uporabnika z brskalnikom. Kot vemo, uporabnikova komunikacija ne poteka preko protokolov in njegov dostop do spletnega strežnika ni neposreden in poteka preko vmesnega manjkajočega člena - spletnega brskalnika. Deluje kot vmesnik med uporabnikom in strežnikom ter upravlja z njuno komunikacijo. Skrbi za pravilen prikaz vsebine komponente pogleda. S pomočjo kontrolnega dela nadzira uporabnikovo delovanje in pošilja ustrezne zahtevke na strežnik. Lahko bi rekli, da deluje po principu *MVC* [18].



Slika 4: Prikaz arhitekture *MVC* aplikacij *RIA*

Glavni cilj arhitekture *RIA* je izogibanje stalnim obhodom na strežnik in posledičnim prekinitvam uporabnikovega dela. Iz tega razloga imajo aplikacije *RIA* drugačno arhitekturo od običajnih spletnih aplikacij (Slika 4). Primerjava obeh shem *MVC* razkrije, da imamo pri aplikacijah *RIA* na odjemalčevi strani dodano tudi komponento modela. Ta onemogoča, da bi vsaka uporabnikova interakcija posledično pomenila nov zahtevek na strežnik. S tem povečamo del interakcij, ki zahtevano spremembo obravnavajo in izvedejo kar na strani odjemalca, npr. preverjanje vnosa podatkov, upravljanje s tabelami (razvrščanje, brisanje elementov, itd.). Zaradi tega postane aplikacija odzivnejša in njeno delovanje hitrejše. Če se pojavi potreba po pošiljanju zahtevka na strežnik, se ti izvršijo asinhrono, to je s pomočjo nadzorne komponente na odjemalčevi strani, brez nepotrebnega prekinjanja uporabnikovega delovnega toka.

Naslednja razlika je v komponenti pogleda, ki se nahaja le na odjemalčevi strani. Torej imamo na strežnikovi strani komponento modela, na katero vplivajo spremembe ob prejeti zahtevi odjemalca. Spremembe komponente pogleda se izključno izvršijo na odjemalčevi strani v spletnem brskalniku. Posledica te spremembe je, da strežnik na tako zahtevo ne odgovori s pošiljanjem celotne strani v obliki dokumenta *HTML*, ampak vrne le posodobljeno komponento modela v obliki podatkov *XML, JSON*, v navadnem tekstu ali kakšni drugi obliki.

3 AJAX – PLATFORMA ZA RAZVOJ APLIKACIJ RIA

V tem poglavju bomo spoznali tradicionalni in Ajax aplikacijsko-komunikacijski model spletnih aplikacij in prikazali razlike v njunem delovanju. Izpostavljen bo predvsem tehnični vidik izvedbe pristopa Ajax. Podrobno bomo spoznali osnovni način dajanja zahtevkov in obravnavanja odgovorov, uporabljen na začetku, ko uporaba orodjarn Javascript še ni bila prisotna v taki meri, kot je danes.

3.1 Uvedba pristopa Ajax

S prihodom že omenjene nove generacije svetovnega spleta in njegovega hitrega razvoja je bil omogočen tudi razmah pristopa Ajax. Na njem temelji razvoj novega roda spletnih aplikacij, ki povezujejo različne tehnologije, in spletni standardi. Ti se združujejo z osnovno idejo o spletu (spletnem brskalniku) kot platformi za njihov razvoj in izvajanje [9]. Med letoma 2001 in 2004 je njihov napredek nekoliko zamrl. Ideja o spletu kot platformi pa je ponovno zaživela šele po tem obdobju. Od takrat do danes smo priča velikemu porastu spletnih aplikacij, ki hočejo biti del propagande (novega roda spletnih aplikacij) s tem, da označujejo svoje aplikacije z izrazom »*Web 2.0*«. Tu prihaja do očitne zlorabe, saj želi vsak potisniti svoje rešitve med vodilne in najuspešnejše spletne aplikacije. Te so opredeljene z lastnostmi :

- močna udeležba uporabnikov kot vir podatkov,
- uporaba kolektivne inteligence,
- enostavni in naprednejši uporabniški vmesniki,
- uporaba razvojnih in poslovnih modelov,
- uporaba arhitekture, ki omogoča izvajanje aplikacije kot spletnega servisa.

Gledano s tehnološkega vidika, je pristop Ajax tisti, ki je povzročil preskok pri prehodu v novo obdobje svetovnega spleta. Kot nov koncept pri razvoju spletnih aplikacij temelji na tehnologijah, poznanih že iz preteklosti. Med seboj povezuje naslednje spletne tehnologije:

- predstavitev podatkov z jezikom *(x)HTML* in *CSS*,
- dinamičen prikaz in interakcija s pomočjo tehnologije *DOM*,
- upravljanje in izmenjava podatkov s pomočjo jezika *XML* in *JSON*,
- asinhrono nalaganje podatkov s pomočjo pogona Ajax,
- jezik Javascript za povezovanje vseh že naštetih tehnologij.

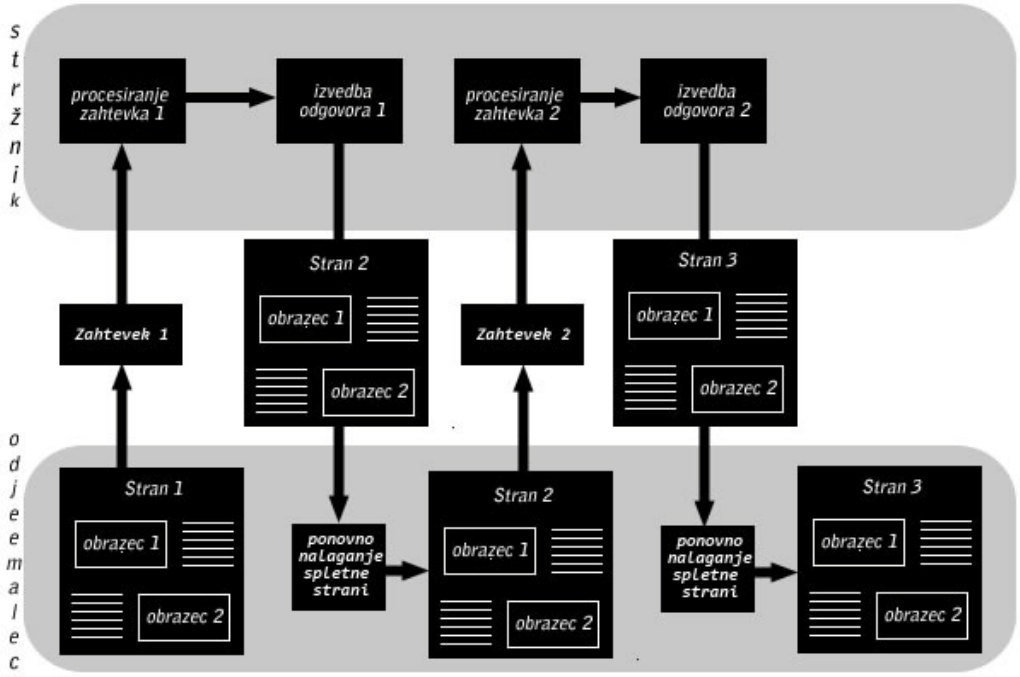
Omenjene spletne tehnologije delujejo, tako rekoč, že v vsakem sodobnejšem spletnem brskalniku. Njihov pomemben del je pogon Ajax, ki skrbi za asinhrona osveževanja delčkov vsebine spletnih aplikacij. Komunikacija uporabnika preko uporabniškega vmesnika aplikacije in komunikacija pogona Ajax s strežnikom se tako lahko izvajata vzporedno in neodvisno druga od druge. Posledica tega so manjši odzivni časi, kar daje uporabniku občutek hitrega odzivanja na podane zahtevo.

3.2 Aplikacijsko-komunikacijski model

Z uvedbo tehnologije Ajax se je spremenil tudi komunikacijski model spletnih aplikacij. Ta ne deluje več po principu *zahtevek/odgovor*, ki ga poznamo iz tradicionalnih spletnih aplikacij, ampak uvaja tehniko delnega osveževanja s pomočjo pogona Ajax. Ker poteka sedanji razvoj v smeri vedno bolj naprednih spletnih aplikacij, prihaja vse bolj do izraza uporaba tega modela, saj vidno izboljša učinkovitost delovanja aplikacije. Tradicionalni model ostaja primeren za uporabo pri enostavnejših spletnih straneh.

3.2.1 Tradicionalni komunikacijski model spletnih aplikacij

Tradicionalni model izvajanja komunikacij poteka na sinhron način, po principu *zahtevek/odgovor* (Slika 5). To pomeni, da vsaki zahtevi uporabnika sledi njeno procesiranje na strežniški strani. Ves ta čas mora uporabnik čakati na vrnjen odgovor in je priča pogostim čakalnim dobam (navadno po vsakem zahtevku). Šele po zaključku celotnega cikla *zahtevek/odgovor* ima uporabnik možnost izvršiti naslednji zahtevek. Njihovo izvajanje poteka zaporedno, eden za drugim, in ne omogoča vzporednega izvajanja. Uporaba takega modela se v aplikacijah *RIA* izkaže za manj zmogljivega in ne daje občutka tiste odzivnosti, ki smo je navajeni pri namiznih aplikacijah.



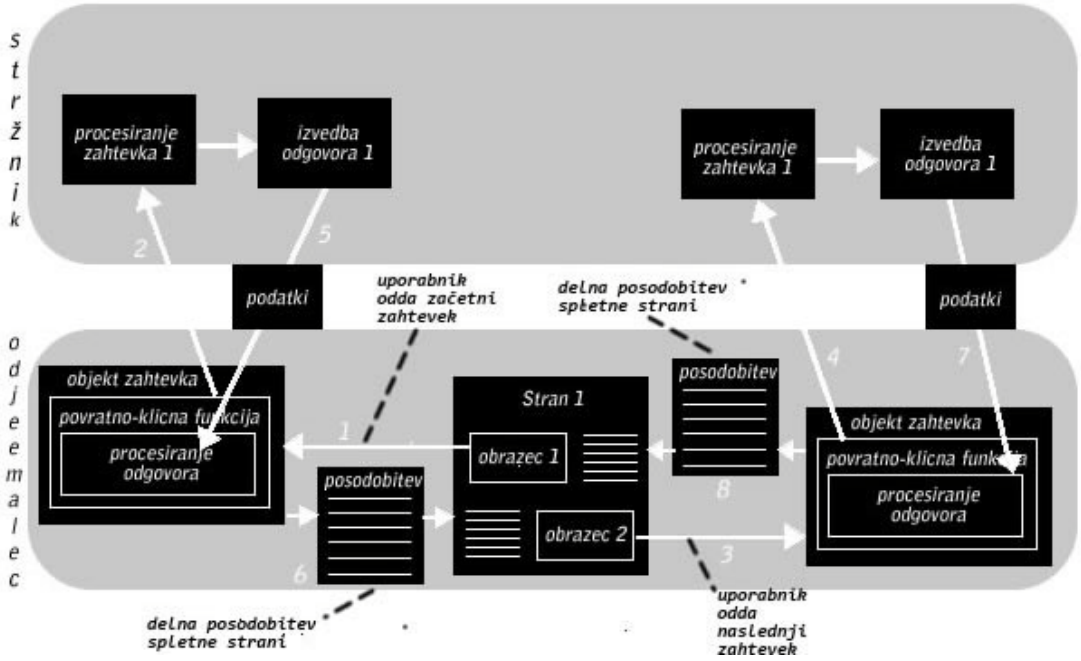
Slika 5: Potek komunikacije v tradicionalnem komunikacijskem modelu

Celoten cikel poteka komunikacije takega modela med odjemalcem in strežnikom si lahko, za lažjo predstavitev delovanja, zamislimo primer, ki uporablja obrazec za registracijo uporabnika. Postopek se začne z izpolnitvijo polj na obrazcu (korak 1). Spletni brskalnik izvrši uporabnikov zahtevek na strežnik, ki zahtevo prejme in jo obdela (korak 2). Vrne ustrezen odgovor v obliki spletne strani, ki se prikaže uporabniku (korak3). Pred tem se zgodi začasna prekinitev prikaza, kjer mora uporabnik čakati na odgovor strežnika ves čas

obdelovanja zahtevka. Šele potem se izvrši prikaz nove predstavitve (korak 4). V primeru novega zahtevka se celoten postopek ponovi.

3.2.2 Komunikacijski model Ajax

Komunikacijski model Ajax vsebuje dodan vmesni nivo med odjemalcem in strežnikom, ki skrbi za upravljanje komunikacije med njima in zajema pogon Ajax ter njegov pomemben del, objekt *XHR*. Skupaj tvorita orodje za uspešno izvedbo asinhronih komunikacij v spletnem brskalniku (Slika 6).



Slika 6: Potek komunikacije v komunikacijskem modelu Ajax

Celoten cikel izvedbe asinhrone komunikacije lahko poteka s hkratnim izvajanje več zahtevkov. Najprej je ustvarjen objekt *XHR*, ki služi upravljanju s prejetimi zahtevami uporabnika (korak 1) in njihovim pošiljanjem na strežnik (korak 2), kjer se obdela in vrne ustrezen odgovor. Zahtevki se pošiljajo asinhrono, tako da ostaja uporabnik v interakciji z aplikacijo med izvajanjem zahtevka na strežniku. Delovni tok uporabnika se ne prekinja, poteka tekoče čez celoten cikel izvajanja zahtevka. V tem času so lahko podani še dodatni zahtevki, ki se obravnavajo vzporedno s predhodnimi (korak 3, 4). Ko strežnik odgovori na prvotni zahtevek (korak 5) objekta *XHR*, se izvrši klic funkcije na odjemalčevi strani. Ta je imenovana povratna klicna funkcija[7]. Vrnjene podatke s strežnika se ustrezno obdela in posodobi del vsebine spletne strani (korak 6). V istem trenutku se lahko strežnik odzove še na druge zahtevke (korak 7) in tako se na odjemalčevi strani izvedejo še druga delna posodabljanja vsebine spletne strani (korak 8) [10].

Predstavljen način komunikacije, ki ga uvaja pristop Ajax, bistveno zmanjšuje količino prenesenih podatkov in obremenitev prenosnega kanala. Če tako zasnovana spletna aplikacija

[7] Funkcija, ki se izvrši ob prejetem odgovoru s strežnika, ko so pridobljeni podatki na voljo.

uporablja relativno majhne (po količini podatkov) zahtevke, glede na širokopasovno povezavo lahko dosežemo večjo odzivnost in pospešimo njeno delovanje. To daje, ob njeni uporabi, podoben občutek, kot smo ga vajeni pri namiznih aplikacij.

3.3 Tehnični pregled pristopa Ajax

V tem delu so predstavljene tehnologije, s katerimi se zagotovo sreča vsak razvijalec, katerega namen je uporabljati pristop Ajax. Njihov nivo predstavitve ne sega v podrobnosti, ampak je predstavljen do te mere, da bralec dobi osnovno vedenje in si s tem ustvari popolnejšo sliko delovanja posameznih tehnologij in celote. Bolj nadzorno bo predstavljen objekt *XHR* kot pomembnejši del pristopa Ajax.

3.3.1 Bistvene tehnologije (spletni standardi) pristopa Ajax

Spajanje različnih tehnologij v pristopu Ajax je že poznano, njihovo razumevanje pa je ključno, v kolikor želimo zagotoviti njihov skladni razvoj s spletnimi standardi. Cilj razvijalca je, da zagotovi enako izvajanje aplikacije v različnih spletnih brskalnikih kljub njihovi različni interpretaciji. Tu gre za tehnologije, uporabljene pri delu na predstavitveni ravni (*(x)HTML, CSS, DOM*) in njihovega povezovanja z jezikom Javascript. Poseben poudarek je namenjen manj poznanemu objektu *XHR* za upravljanje z asinhronimi zahtevki in različnim oblikam podatkov za njihov prenos (*XML, JSON*).

Jezik Javascript

Znotraj pristopa Ajax je jezik Javascript osrednji element za povezovanje vseh ostalih tehnologij. Ta povezuje različne prikaze in logiko aplikacije na odjemalčevi strani. Ima sposobnost preoblikovanja dokumenta *HTML* in v ta namen izkorišča vmesnik *DOM*. Je programski jezik s podobnimi lastnostmi (kot jeziki družine C programskih jezikov):

- ***splošno namenska uporaba*** - primeren je za izdelavo različnih algoritmov in izvedbo programskih nalog;
- ***ohlapnejše določena sintaksa*** - spremenljivke niso deklarirane z določenim tipom. Priredimo jim lahko vrednosti, različnih tipov (niz znakov, število, objekt, itd.);
- ***interpretiran (tolmačen) jezik*** - izvorna koda se ne prevede v izvršno, ampak se prenese s strežnika na odjemalca, kjer se v brskalniku neposredno izvede;
- ***objektno usmerjen jezik*** - ne uporablja vseh principov *OOP-ja*, kot jih poznamo pri nekaterih drugih programskih jezikih, vendar jih poskuša simulirati na drugačne načine. Plod tega je več različnih vzorcev dedovanja, s katerim povečamo uporabo že razvitih delov programske kode. To uvaja nekatere, ne tako poznane, lastnosti programskih jezikov, npr. uporaba prototipov, funkcijsko programiranje.

Jezik Javascript ni bil zasnovan za izdelavo obsežnih aplikacij, ampak za pisanje manjših programov za izvedbo določenih funkcionalnosti. V začetku se je uporabljal samo vgrajen znotraj posameznega dokumenta *HTML* kot enostavna, nekaj vrstična programska koda. Danes je način uporabe drugačen. To se kaže predvsem pri spletnih *(»web 2.0«)* aplikacijah, kjer se njihovo delovanje v veliki meri zanaša prav na jezik JavaScrip. To je opazno predvsem

pri izvajanju na odjemalčevi strani (uporabniškega vmesnika). Pri tem opazimo večje količine programske kode Javascript, ki mora biti prenesena ob začetnem zagonu spletne aplikacije. Njen prenos s strežniške na odjemalčevo stran lahko poteka v nespremenjeni obliki zaradi potrebe tolmačenja. Jezik Javascript je tudi osnova, na kateri temelji delovanje večine orodjarn Javascript.

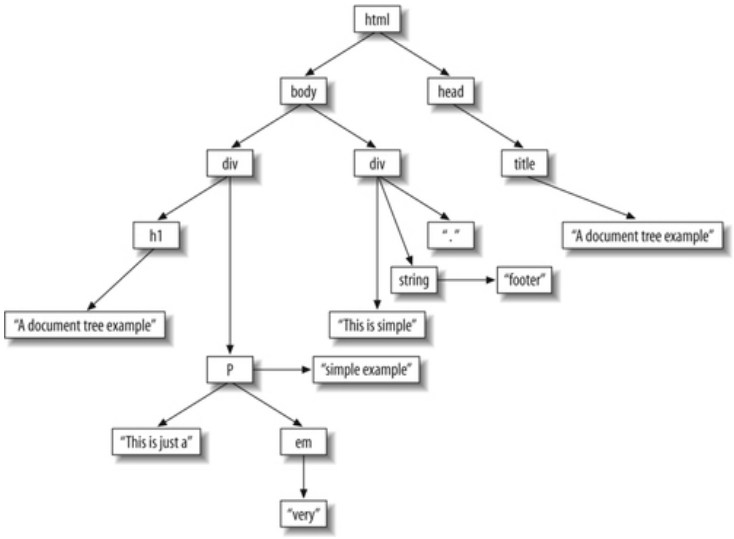
Poznavanje jezika Javascript in vseh njegovih sposobnosti je pomembno, če želimo izkoristiti njegovo moč pri povezovanju vseh ostalih tehnologij v okviru pristopa Ajax.

Objekt XMLHttpRequest (XHR)

Objekt *XMLHttpRequest* ima pomembno vlogo pri razvoju spletnih aplikacij, ki temeljijo na pristopu Ajax. Njegova naloga je pošiljanje zahtevkov (v ozadju na asinhron način) in prejemanje odgovorov s pomočjo jezika Javascript. Odgovore lahko prejme v različnih oblikah, ki souporabljeni pri manipulaciji trenutno aktivnega dokumenta v brskalniku.

Dokumentni objektni model (DOM)

Dokumentni objektni model predstavlja elemente dokumentov *XML* in *(x)HTML*, do katerih lahko dostopamo z uporabo jezika Javascript. Največkrat je ta struktura predstavljena v drevesni obliki (Slika 7). Zgrajena je iz posameznih objektov, ki predstavljajo gradnike spletne strani. Lahko bi rekli, da povezava med elementi v strukturi *DOM* zrcali stanje elementov *HTML* spletne strani. Povezava med njimi je dvosmerna. S spremembo strukture *DOM* se spremeni tudi struktura *HTML* in prikaz strani.



Slika 7: Drevesna struktura *DOM* dokumenta *HTML*

V povezani celoti drevesne strukture nam za izhodiščno točko pri upravljanju služi globalna spremenljivka »*document*«. Z njeno pomočjo lahko dostopamo do poljubnega elementa v strukturi in ga programsko spreminjamo s pomočjo jezika Javascript in pravil, ki jih določa aplikacijski programski vmesnik *DOM*. Ta lastnost, ki jo uporabljamo pri aplikacijah Ajax, je koristna za delne spremembe uporabniškega vmesnika.

Element v strukturi *DOM* je določen z enim nadrejenim elementom in nič ali več podrejenimi. Vsebuje poljubno število atributov, ki so shranjeni v asociativnih tabelah.

V naslednjem poglavju bomo spoznali, so se nekatere orodjarne Javascript specializirale prav za izvedbo koristnih funkcij, ki olajšajo delo s tehniko *DOM*. Te so v veliki meri v pomoč razvijalcu, ki mu predstavlja manipulacija z elementi strukture *DOM*, z uporabo orodjarn Javascript, veliko lažje opravilo.

Prekrivni slogi (CSS)

Prekrivni slogi so jezik, ki ga uporabljamo za opis predstavitve (izgleda in oblike) spletnega dokumenta, napisanega v označevalnem jeziku. Tipi dokumentov so lahko *(x)HTML, XML* in drugi.

Jezik *CSS* je bil primarno zasnovan z namenom ločiti vsebino od predstavitve (barve, pisave, slike, postavitev elementov, itd.) dokumenta. To zagotavlja večjo fleksibilnost, zmanjšuje kompleksnost ter omogoča boljšo dostopnost za njihovo upravljanje [03]. Sintaksa sestoji iz številnih pravil, s pomočjo katerih določamo slog elementov. Pravilo je sestavljeno iz dveh delov. Najprej moramo določiti *selektor,* nato še njegovo deklaracijo. Z njim določimo, kateri element bomo stilno uredili, njegova deklaracija pa nam pove, katere stilne lastnosti bodo uporabljene. Skupek tako določenih pravil lahko hranimo v ločeni datoteki ali vstavljene znotraj dokumenta *HTML*. Zaradi različne interpretacije pravil v posameznih spletnih brskalnikih je pomembna naloga razvijalca, da zmanjša te razlike in v vseh spletnih brskalnikih (tudi starejših različicah) zagotovi čim bolj podoben izgled spletne aplikacije.

Ob osnovni uporabi prekrivnih slogov se ti v aplikacijah Ajax uporabljajo za zagotavljanje povratne informacije uporabniku preko uporabniškega vmesnika, z npr. prikazom indikatorja, da gre za obdelovanje zahtevka; s prikazom določenih elementov spletne aplikacije, na katere ima uporabnik vpliv, npr. ob prehodu čez element z miškinim kazalcem se spremenijo stili elementa.

Upravljanje s prekrivnimi slogi tvori širšo opredelitev pristopa Ajax, ki pa ni zajeta v samem poimenovanju. Zato ti niso nič manj pomembni od ostalih tehnologij, ki so združene pod tem pristopom. Na to kaže potreba po različnih vizualnih učinkih, stilnih predlogah, animacijah, kontrolah, npr. povleci in spusti …

Razširjeni označevalni jezik ((x)HTML)

Razširjeni označevalni jezik je sredstvo za oblikovanje strukture spletnih dokumentov s pomočjo elementov *HTML* kot osnovne sestavine. Vsebujejo dve osnovni lastnosti, atribute in vsebino. Za posamezen atribut in vsebino elementa veljajo določene omejitve, ki morajo biti upoštevane, če hočemo zagotoviti veljavnost dokumenta. Znotraj tako sestavljenega dokumenta lahko vključimo datoteke, že prej omenjenih tehnologiji jezika Javascript in jezika *CSS*.

Oblika prenosa podatkov XML in JSON

Oblika prenosa podatkov *XML* in *JSON* se uporabljata za prenos podatkov v aplikacijah Ajax. Ta sta pogosta izbira razvijalcev, vendar poznamo, poleg njiju, še druge oblike, ki se prav tako uporabljajo v ta namen. Na izbiro primerne oblike podatkov v aplikacijah Ajax primarno vplivajo naslednji trije faktorji:

- ***enostavnost kodiranja in dekodiranja***. Pri aplikacijah Ajax je izbira oblike za pošiljanje podatkov v razvijalčevih rokah. Pri tem se lahko odloča za običajno obliko (par *ime/vrednost*) ali poseže po kateri izmed možnih alternativ, npr. obliko pošiljanja v formatu *JSON* ali v dobro poznanem formatu *XML*. Taka izbira ima lahko za posledico nezaželene učinke, saj je potrebno zagotoviti dekodiranje takega formata na strežniški strani. Posledično je težje tudi njihovo razčlenjevanje;
- ***prednosti z vidika varnosti.*** Omogoča tak prenos podatkov, ki ni preprosto človeško berljiv. Najboljša rešitev je seveda kriptiranje prenosa preko protokola *SSL;*
- ***učinkovitost pri prenosu.*** Nekateri formati so lahko precej potratni, če primerjamo njihovo vsebino s strukturo. Drugi potrebujejo za pošiljanje podatkov veliko manj strukturnih elementov glede na količino vsebine.

Oblika podatkov *XML* je zajeta v poimenovanju Ajax, vendar se ne uporablja več v tolikšni meri za njihov prenos. Običajno jo srečamo pri razčlenjevanju prejetega odgovora, ki ga prejmemo na podan zahtevek. Razlog tega je najverjetneje v težavnosti pri kodiranju in dekodiranju. Morda je prav to eden od razlogov za uveljavitev formata *JSON*, ki se v zadnjem času vse bolj uporablja. Temelji na jeziku Javascript, zato je tudi njegovo poimenovanje temu primerno - Javascript objektna notacija. Kljub temu je jezikovno neodvisen in ga z lahkoto uporabljamo z različnimi programskimi jeziki. Pri prenosu podatkov pride do izraza predvsem v primeru, ko je potrebno zmanjšati velikost zahtevkov. To zagotavlja dobra strukturiranost podatkov. Primer (1) prikazuje, kako bi bili videti podatki v obliki *JSON*.

```
{"ime": "vrednost",
 "ime_1": [
      "tabela",
      "elementov",
      ". . ."
 ]
}
```

(1)

3.3.2 Komunikacijske tehnike

Vemo, da veliko zahtevkov poteka iz spletnega brskalnika v proti strežniku in nazaj. V prvi vrsti so ti zahtevki posledica uporabnikovega delovanja. Pri uporabi pristopa Ajax razvijalci niso odvisni od podanih zahtevkov uporabnika, saj lahko izvedejo klic na strežnik v poljubnem trenutku. V ta namen so bile razvite različne komunikacijske tehnike, od tistih starejši do novejših. Nekateri menijo, da se pristop Ajax ne nanaša na vse, ki omogočajo delno osveževanje vsebine (npr. tehnika »*iframe*«), vendar ga večina povezuje z objektom *XHR* oziroma v začetku z *Microsoft-ov* kontrolo *XMLHttp Activex*.

Microsoft uvede s svojim brskalnikom Internet Explorer osnovni nivo podpore standardu *XML* in z njim tudi *ActiveX* knjižnico, poimenovano *MSXML*. Eden izmed njenih objektov (*XMLHttp*) je postal hitro priljubljen. Ker je bil objekt *XMLHttp* v začetku zasnovan kot ActiveX kontrolnik, je omogočal uporabo tudi v namiznih aplikacijah (tiste, ki so temeljile na operacijskih sistemih Windows). Njegova priljubljenost se je povečevala in še posebej

zaživela na področju spletnih aplikacij. To so začutili tudi v drugih podjetjih, ki ponujajo spletne brskalnike. Postopoma so ga tudi sami implementiral v svojih brskalnikih kot objekt *XHR*. Ta služi povsem istemu namenu kot že omenjeni kontrolnik *XMLHttp*, vendar je izveden na drugačen način in je poznan pod drugim imenom.

3.3.3 Objekt XMLHttpReguest – izvedba in delovanje

Uporaba objekta *XHR* je najbolj razširjen način za izvedbo komunikacije preko protokola *HTTP* pri pristopu Ajax. Izkaže se predvsem pri uporabi na potencialno dvosmerni komunikaciji. Tu nam služi kot prenosni objekt, znotraj katerega prejmemo vse pomembnejše podatke, skupaj z odgovorom na podani zahtevek. V tem razdelku je prikazan način kreiranja objekta, pošiljanja zahtevka in obdelava prejetega odgovora.

Kreiranje objekta XMLHttpRequest

Iz podpoglavja o komunikacijskih tehnikah smo izvedeli, da obstaja različna implementacija in podpora objekta *XHR* med spletnimi brskalniki. Večina novejših (*Mozzila FireFox, Safari, Opera*) podpira navaden objekt *XHR*. Delijo si enako sintakso, s katero se ustvari primerek takega objekta. To storimo na sledeči način:

```
var xhr = new XMLHttpRequest();                                    (2)
```

To še ne zagotavlja pravilnega delovanja objekta *XHR* za vse brskalnike, predvsem mislimo različice brskalnika *Internet Explorer,* pri katerih je objekt *XHR* izveden kot *ActiveX* kontrolnik. Primerek takega objekta izvršimo z uporabo konstruktorja *ActiveXObject*:

```
var xhr = new ActiveXObject ("Msxml2.XMLHTTP");                    (3)
```

Ker različni brskalniki implementirajo objekt *XHR* na različne načine, moramo ustvariti tak primerek objekta, ki bo ustrezal trenutno uporabljenemu brskalniku. V tem primeru uporabimo tehniko zaznavanja objektov, s katero poskušamo ugotoviti sposobnost brskalnika glede kreiranja različnih objektov. Običajno se oba prejšnja koraka (2 in 3) združi in uporabi v funkciji (4), ki preverja razpoložljivost objekta *XHR* [15].

```
function podpora_XHR() {
     var xhr;
     if (window.XMLHttpRequest) {
          xhr = new XMLHttpRequest();
     }
     else if (window.ActiveXObject) {
          xhr = new ActiveXObject("Msxml2.XMLHTTP");
     }
     else {
          throw new Error("Ajax ni podprt");
     }
}
                                                                   (4)
```

Ko imamo ustvarjen primerek objekta *XHR*, lahko z njim začnemo pošiljati zahteve. V ta namen uporabljamo množico lastnosti in metod, ki jih ta objekt vsebuje, ne glede na vrsto spletnega brskalnika, v katerem je bil ustvarjen. Glavne lastnosti in metode, potrebne za izvedbo zahtevkov, bomo spoznali v tem poglavju.

Problem združljivosti objekta *XHR* z različnimi brskalniki rešuje uporaba Javascript orodjarn. Njihove funkcije za delo z Ajax-om že implementirajo potrebno logiko, ki zagotavlja delovanje objekta *XHR* v različnih spletnih brskalnikih. Primer kreiranja objekta *XHR* na tak način je predstavljen v naslednjem poglavju, kjer so prikazane tudi funkcije za delo z Ajax-om dveh orodjarn Javascript.

Pošiljanje zahtevka s pomočjo objekta XMLHttpRequest

Prvi del v celotnem ciklu izvajanja zahtevka se prične z njegovim pošiljanjem, temu sledi prejemanje odgovora in njegovo razčlenjevanje. Še pred pošiljanjem zahtevka je potrebno nastaviti določene njegove parametre. Ti zajemajo korake, v katerih je potrebno[09]:

−določiti metodo *HTTP* (kot sta *POST* ali *GET*);

−zagotoviti naslov *URL* vira, na katerega bo zahtevek poslan;

−določiti lastnost objekta *XHR*, ki nas obvešča o stanju zahtevka;

−določiti morebitne podatke, ki jih želimo poslati v zahtevi.

Prva dva koraka izvedemo s pomočjo metode *open()* objekta *XHR* naslednji način:

```
//sintaksa
open(http_metoda, internetni_naslov, asinhron, [up_ime, geslo])

//primer uporabe
xhr.open('GET', 'ajax.php?parameter='vrednost'', true);
```
(5)

V tem trenutku še ni poslan zahtevek, ampak le nastavljen spletni naslov in metoda *HTTP*. Metodi *open()* bi lahko podali tudi parameter *asinhron.* Z njim določimo, ali je zahtevek poslan sinhrono (če je *false*) ali asinhrono (če je *true*). Privzeti način izvrševanja zahtevkov je vedno nastavljen na asinhron način, zato je vrednost tega parametra nastavljena na *true*.

V tretjem koraku določimo lastnosti *onreadystatechange* objekta *XHR* funkcijo, ki se izvrši na različnih stopnjah med obdelovanjem zahtevka. Z njeno pomočjo in ob uporabi še ostalih lastnosti objekta *XHR* lahko natančno vemo, kaj se dogaja z zahtevkom.

V zadnjem koraku dodamo podatke, ki jih želimo poslati z zahtevo po metodi *POST*. Pri uporabi metode *GET* podatkov v telesu, zahteve ne pošiljamo. Te lahko določimo kar v naslovu *URL*. V obeh primerih, ne glede na uporabljeno metodo *HTTP*, uporabimo pri pošiljanju zahtevka metodo *send()* objekta *XHR* naslednji način:

```
//za metodo GET
xhr.send(null);
//za metodo POST
xhr.send('parameter=vrednost&parameter1=vrednost1');
```
(6)

Ko sprožimo pošiljanje zahtevka (6), bo funkcija, ki smo ji določili lastnosti objekta *XHR*, poklicana večkrat med procesiranjem tega zahtevka. Ta se lahko nahaja v štirih različnih stopnjah. Trenutno stopnjo lahko preverimo s pomočjo lastnosti *readyState*, v kateri se hrani številčna vrednost trenutne stopnje (Tabela 3.1).

Vrednost	Stanje	Opis
0	neinicializirano	metoda *open()* še ni bila poklicana
1	nalaganje	metoda *open()* je bila poklicana
2	naloženo	metoda *send()* je bila poklicana
3	pošiljanje	del podatkov iz odgovora je bil prejet
4	zaključeno	vsi podatki so prejeti

Tabela 3.1: Vrednosti lastnosti *readyState,* ki prikazujejo stanje zahtevka

Vzorec (7), ki preverja stanje zahtevka znotraj funkcije lastnosti *onreadystatechange*, bo pogosto uporabljen.

```
xhr.onreadystatechange = function() {
     if (xhr.readyState == 4) {
          if (xhr.status >= 200 && xhr.status < 300) {
          //uspeh
          }
          else {
          //napaka
          }
     }
}
```
(7)

V tem primeru (7) prezremo vsa stanja, razen zadnjega, zaključenega, ne. Običajno je skoraj vedno tako, da nas zanima stanje zahtevka, ko je ta v celoti zaključil s procesiranjem in so na voljo podatki. V ta namen preverjamo, če lastnost *readySate* vsebuje vrednost štiri, kar nakazuje zaključek zahteve. Ob tem moramo zares ugotoviti, ali je bila zahteva uspešno obravnavana oziroma ali je prišlo do napake. Zaradi tega moramo preverjati še lastnost *status*. Ta vsebuje, v primeru uspešno pridobljenega odgovora, statusno kodo med 200 in 299, v primeru napake pa statusno kodo, ki ustreza vrnjeni napaki (npr. neveljaven spletni naslov – napaka 404, itd.).

Obdelava odgovora objekta XMLHttpRequest

Obdelava odgovora sledi, v primeru uspešno vrnjenega odgovora, s strežnika. Namen tega je ustrezna razčlenitev prejetega odgovora in pridobitev želenih podatkov. Podlago za to smo si zagotovili že prej, da smo objektu *XHR* oziroma njeni lastnosti *onreadystatechange* pripisali funkcijo. Z njeno pomočjo bomo lahko pridobili ustrezne podatke, potrebne pri posodobitvi uporabniškega vmesnika.

Ko s pomočjo lastnosti *onreadystatechange* ugotovimo, da je zahtevek uspešno zaključen, lahko pridobimo podatke iz odgovora. Ne glede na uporabljen format pri pošiljanju zahtevka nam je vsebina odgovora dostopna s pomočjo lastnosti *responseText* (ob predvidevanju, da se je zahtevek uspešno zaključil). Če določa odgovor v glavi tip vsebine

zahtevka, npr. tip *MIME* z vrednostjo *text/xml*, bo odgovor razčlenjen kot *XML*. V tem primeru bo rezultat odgovora shranjen v drugi spremenljivki objekta *XHR*, poimenovani *responseXML*.

Prejete podatke lahko uporabimo in z njimi posodobimo posamezne dele dokumenta. To je tudi tipična naloga, ki jo opravimo s pomočjo Ajax-a. Če združimo programsko logiko prikazanih primerov tega razdelka in dodamo še malo nove, lahko zapišemo primer (8) za posodabljanje elementa z določenim identifikatorjem.

```
var xhr;
if (window.XMLHttpRequest) {
     xhr = new XMLHttpRequest();
}
else if (window.ActiveXObject) {
     xhr = new ActiveXObject("Msxml2.XMLHTTP");
}
else {
     throw new Error("Ajax ni potprt");
}

xhr.onreadystatechange = function() {
     if (xhr.readyState == 4) {
          if (xhr.status >= 200 && xhr.status < 300) {
               document.getElementById('elementDiv').innerHTML =
          xhr.responseText;
          }
     }
}

xhr.open('GET','/strezniskiVir');
xhr.send();
```

(8)

Lahko vidimo, da smo zapisali veliko vrstic programske logike, čeprav ne gre za nič zapletenega. Ekvivalenten primer, ki bi ga zapisali s poljubno orodjarno Javascript, bi obsegal le nekaj vrstic. Orodjarne Javascript ponujajo tudi številne izboljšave in tako odpravljajo težave, s katerimi se soočamo pri uporabi navadnega jezika Javascript. V preteklosti je bila uporaba objekta *XHR* na način, kot je predstavljen v tem poglavju, edina možnost za pošiljanje asinhronih zahtevkov. Danes, ko poznamo množico različnih orodjarn Javascript, se zdi njihova uporaba bolj smiselna, saj nam prihranijo veliko časa pri razvoju obogatenih spletnih aplikacij.

4 ORODJARNE (AJAX) JAVASCRIPT

Izrazi, ki se vzporedno pojavljajo na tem področju z orodjarnami, so še ogrodja in knjižnice. Njihovi koncepti so v osnovi enaki in se medsebojno prepletajo. Med njimi obstoje določene razlike, a kljub njihovi različni opredelitvi, so pogosto uporabljeni kot sinonimi. Zaradi nebistvenih razlikovanj med posameznimi koncepti bomo v tej nalogi uporabljali izraz orodjarne. Pomembnejše od izbire ustreznega poimenovanja so njihove lastnosti in karakteristike.

Uporabnikova pričakovanja so pripeljala razvoj aplikacij, ki temeljijo na pristopu Ajax, do te mere, da so začele ponujati drugačen koncept delovanja uporabniškega vmesnika. Do pred nekaj let je bilo uresničevanje tega z jezikom Javascript neprijetno in posledično izogibajoče se opravilo. Danes se je z uvedbo orodjarn Javascript vse to spremenilo. Njihova ideja je v abstrakciji napisane programske kode za različne pogone spletnih brskalnikov, v katerih se izvajajo. To pomaga razvijalcem premagati najbolj frustrirajoč del pri delovanju jezika Javascript, okolje za njegovo izvajanje – spletni brskalnik – je del, nad katerem razvijalec nima nadzora, in njihovo obnašanje se razlikuje zaradi različne implementacije konkurenčnih standardov.

Orodjarne Javascript blažijo težave razvijalca pri uporabi *»surovega«* objekta *XHR*, s katerim se srečamo pri razvoju aplikacij, ki uporabljajo pristop Ajax. Tu mislimo predvsem na probleme pri pošiljanju asinhronih zahtevkov na strežnik, prejemanje in procesiranje njihovih odgovorov ter spreminjanje delov spletne strani glede na novo pridobljene podatke. Prinašajo tudi celo vrsto drugih prednosti, med katerimi so najpomembnejše:

- abstrakcija izvorne kode okolja za izvajanje (brskalnika) tako, da so lahko različna nepredvidena obnašanja obravnavana s pomočjo orodjarne, upravljanje z njimi poteka na ene mestu;
- z osnovno programsko kodo je postavljen temelj, na katerem se lahko gradi dodatne funkcionalnosti, kar omogoča enoten razvoj aplikacije;
- razvoj temelji na odprti kodi, tako je vzpostavljeno skupno okolje, v katerega lahko prispevajo različne skupine in posamezniki (svoje lastne razširitve, vtičnike, popravke, itd.);
- spodbujanje k uporabi podobnih vzorcev za različne splete aplikacije, kar omogoča uporabniku ponovno uporabo že predhodno pridobljenega znanja.

Omogočajo uporabo tudi različnih, že izdelanih gradnikov, kot na primer nove elemente uporabniškega vmesnika (npr. drevesni pregledi, drsni gumbi, kontrola povleci in spusti, itd.) za zagotavljanje bogatejše uporabniške izkušnje, kot tudi možnost izdelave lastnih. Ker so določene funkcionalnosti potrebne že v vsaki spletni aplikaciji, ki uporablja Ajax, podpora z uporabo orodjarn Javascript precej olajša njihovo izvedbo.

V tem trenutku je veliko podjetij, ki se odločajo za izbiro ustrezne orodjarne Javascript oziroma se bodo s tem problemom soočile v bližnji prihodnosti. Odločitev o ustrezno izbrani orodjarni je lahko ključnega pomena za uspešno zaključen projekt. Njihova naloga bo tako

podobna naši, kjer bomo v sledečih razdelkih razpravljali o lastnostih, ki jih morajo zagotoviti orodjarne Javascript. Govorili bomo o tistih, ki so pomembnejše in bolj koristijo razvijalcu, in tudi o tistih, ki so zaželene in jih lahko razvijalec uporabi. V nadaljevanju bodo predstavljene različne kategorije, v katere lahko razvrstimo posamezne orodjarne. Nato se bomo osredotočili le na eno skupino orodjarn in predstavili konkretno primerjavo dveh izmed njih. To sta orodjarni, ki se izvajata na strani odjemalca in sta pogosto izbiri razvijalcev pri izdelavi različnih spletnih projektov.

4.1 Zahteve za orodjarne Javascript

Veliko je uporabnih stvari, ki jih razvijalec pričakuje, da jih bodo orodjarne Javascript izpolnjevale. Zaradi tega smo seznam njihovih lastnosti razdelili v dve skupini. Bistvene so lastnosti, ki omogočajo izvedbo osnovnih funkcionalnosti in zaželene lastnosti ter omogočajo razvijalcu uporabo dodatnih funkcionalnosti.

Osnovne lastnosti so vitalnega pomena in njihov nivo implementacije določa uporabnost orodjarne. Zagotovo obstajajo tudi take orodjarne, ki ne ustrezajo vsem osnovnim zahtevam (lastnostim) oziroma jih implementirajo do določene mere. Zaradi tega niso nič manj uporabne, saj lahko v nekem trenutku pridejo ravno te najbolj do izraza. V takih primerih je to odvisno predvsem od projekta, ki ga z izbrano orodjarno želimo podpreti. Sledeči seznam lastnosti orodjarn vsebuje tiste osnovne, ki naj bi jih v večini vsebovale vse orodjarne:

- ***upravljanje z asinhronimi zahtevki.*** Orodjarna mora zagotavljati ustrezen način reševanja problemov, kot so: pošiljanje asinhronih zahtevkov, prejemanje strežniškega odgovora, posodabljanje vsebine elementov *DOM* s prejetim odgovorom. Ti deli programske kode so ključni in potrebni v vsaki aplikaciji Ajax, zatorej mora biti njihova izvedba vključena v orodjarno, če želimo uporabljati tehniko Ajax;
- ***obravnavanje razlik med spletnimi brskalniki.*** Ker obstajajo nekatere razlike med različnimi spletnimi brskalniki, mora orodjarna zagotoviti enako izvedbo aplikacije na različnih odjemalcih. To vključuje upravljanje z razlikami pri uporabi objekta *XHR* (v nalogi smo že spoznali, da moramo brez uporabe orodjarn sami poskrbeti za pravilno izvedbo objekta *XHR* glede na vrsto uporabljenega spletnega brskalnika), jezika Javascript in manipulacijo s pomočjo tehnike *DOM*;
- ***učinkovito procesiranje podatkov.*** Orodjarna mora zagotoviti metode za enostavno in učinkovito procesiranje podatkov, prejetih s strežnika. To vključuje podatke v obliki *XML, JSON* ali navadnem tekstu. Prav tako mora omogočati enostaven dostop do njih;
- ***naklonjeno rabo sistemskih virov.*** Učinkovitejša raba sistemskih virov orodjarne zagotavlja izvajanje spletne aplikacije tudi v kombinaciji z ostalimi aplikacijami, in to tudi na starejših napravah;

- ***zadostna dokumentacija.*** Dobro napisana dokumentacija orodjarne zagotavlja podroben opis njenega celotnega aplikacijskega programskega vmesnika – *API-ja*[8]. Ta je ključnega pomena, saj omogoča, da se razvijalec v čim krajšem času seznani z načinom delovanja orodjarne in njenimi funkcionalnostmi;
- ***uporaba po principu »črne skrinje«***[9]**.** Uporaba orodjarne ne sme biti pogojena s podrobnim poznavanjem implementacije njenih funkcij. Razvijalcu mora biti omogočeno, da se hitro seznani z delovanjem orodjarne in začne koristiti njene funkcionalnosti v najkrajšem času. Tako ni potrebna pretirana poraba dodatnega časa pri spoznavanju notranjega delovanja orodjarne, če tega razvijalec ne želi;
- ***množico obogatenih elementov uporabniškega vmesnika.*** Orodjarne Javascript naj ne bi zagotavljale samo funkcij za pošiljanje in procesiranje zahtev s pomočjo objekta *XHR*, ampak tudi množico obogateni elementov (tako imenovanih gradnikov), ki omogočajo izboljšavo uporabniške izkušnje. Ti morajo biti nastavljivi na razvijalčevi strani in vključujejo, na primer urejene sezname, drevesne strukture, modalna okna, samodejne dopolnjevalce besed, itd.;
- ***čvrsto ozadje.*** To pomeni, da razvoj poteka orodjarne s strani podjetja ali organizacije, ki obstoja že dlje časa ter redno skrbi za izdajanje novih izboljšav in vzdrževanje orodjarne. Ključnega pomena pri orodjarni je tudi njena stopnja razvitosti, ki je že dosežena. Dober pokazatelj tega je trenutna različica (številka) izdaje orodjarne, datum njene prve javne izdaje in pogostost intervalov, ob katerih se izdajajo nove različice oziroma posodobitve orodjarne;
- ***čas usposabljanja in stroški uvedbe.*** Doba usposabljanja naj bi bila sorazmerna s koristmi, ki jih ponuja orodjarna. Stroški uvedbe orodjarne pa naj bi se amortizirali že po prvem oziroma nekaj projektih, odvisno od njihove velikosti;
- ***izboljšanje uporabnosti obstoječe aplikacije.*** Uporaba orodjarne bi morala pomagati izboljšati delovanje obstoječe aplikacije na učinkovitejši način kot brez njene uporabe. Če mora biti aplikacija v celoti ponovno zasnovana samo zaradi uvedbe funkcionalnosti Ajax, je potrebno uporabo orodjarne pretehtati in oceniti, ali je sploh smiselna.

Seznam zaželenih lastnosti, ki naj bi jih v čim večji meri ponujale orodjarne Javascript, je lahko precej dolg, če se ne omejimo le na najbolj zaželene. Vse zagotovo niso, v celotnem obsegu, prisotne v vsaki orodjarni Javascript. Njihove manjkajoče lastnosti lahko služijo snovalcem orodjarn kot namig pri razvoju njihovih prihodnjih izboljšav. Seznam vključuje naslednje lastnosti orodjarn:

---

[8] Vmesnik uporabniškega programa, ki namenskim programom omogoča klicanje funkcij operacijskega sistema ali drugega računalniškega programa.

[9] Izraz za napravo, sistem ali predmet (v našem primeru gre za funkcije orodjarne), ki ga obravnavamo v smislu njegovih vhodnih in izhodnih značilnosti, brez poznavanja notranjega delovanja.

- ***izvedba v odprti kodi***. Izvedba orodjarne mora biti dostopna razvijalcu do take mere, da mu omogoča podroben pregled njenih funkcij in seznanitev z njihovim podrobnim delovanjem. To je lahko uporabno pri nadaljnjih izboljšavah orodjarne, kot tudi za učenje na podlagi pregleda izvedbe posameznih funkcionalnosti;
- ***ločenost programskih delov.*** Orodjarne Javascript bi morale podpirati princip arhitekturne ločitve različnih delov aplikacije. Predstavitveni, podatkovni in kontrolni modeli morajo biti uporabljeni, če želimo, da spremembe v aplikacijah preveč medsebojno ne vplivajo;
- ***podpora in integracija z orodji***. Dobro je, če obstaja za orodjarno vsaj nekaj integriranih razvojnih okolij (angl. *IDE*) oziroma orodij, ki so v podporo razvojnemu procesu aplikacij. Te vsebujejo že nekatere predhodno izdelane gradnike ali druge elemente aplikacije, orodja za razhroščevanje in testiranje izvorne kode ter druge uporabne instrumente;
- ***nadomestne rešitve. Če*** spletni brskalnik ne podpira asinhronih zahtevkov ali je izvajanje jezika Javascript onesposobljeno od uporabnika, mora orodjarna zagotoviti ustrezno rešitev (npr. dajanje asinhronih zahtevkov s pomočjo medvrstičnih okvirjev – *»iframe«*). Če tega ne zagotavlja, mora o tem obvestiti uporabnika in ponuditi namig za rešitev problema (npr. potrebno je omogočiti izvajanje jezika Javascript, uporabiti drug brskalnik);
- ***skladnost z drugimi orodjarnami.*** Orodjarne Javascript bi morale biti do neke mere združljive z drugimi orodjarnami. V nekaterih primerih bi bilo lahko koristno združiti dve različni orodjarni in koristiti prednosti vsake izmed njiju;
- ***lahko nadomestljive.*** Če orodjarna ni več vzdrževana od proizvajalca oziroma njenega snovalca, je zaželeno, da jo nadomesti z drugo, in sicer brez večje porabe dodatnega časa pri učenju drugačnih principov.

4.2 Kategorizacija orodjarn Javascript

V tem razdelku predstavljamo različne kategorije, v katere lahko razvrstimo posamezne tipe orodjarn Ajax. Na voljo imamo štiri različno opredeljene ravni, med katerimi lahko izbiramo pri prevzemu pristopa Ajax in jih lahko izvedemo v določeni aplikaciji [21]. Pristop Ajax vsebuje repertoar različnih tehnik, zato jih lahko razvijalec uporabi v aplikaciji v poljubnem obsegu (Slika 8).

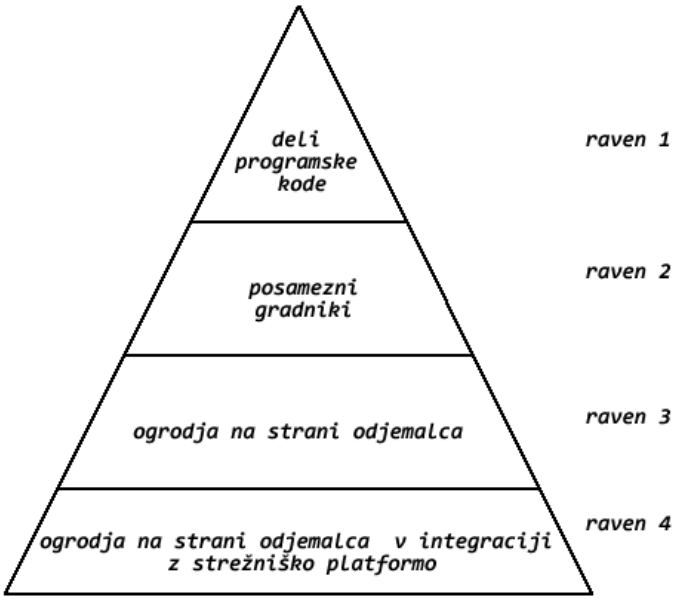
Prva raven implementacije vsebuje le koščke programske kode *(angl. Snippets)*, ki jih lahko dodamo v obstoječo aplikacijo, brez znatnega spreminjanja arhitekture aplikacije. Na ta način lahko z minimalnim vložkom izvedemo določeno koristno spremembo aplikacije. Denimo, dodamo lahko preverjanje poštne številke na spletnem obrazcu, kar nam zagotavlja večjo odzivnost in zmanjša število obhodnih zahtevkov na strežnik ter posledično ponovno nalaganje celotne spletne strani.

Naslednja, druga stopnja, je raven implementacije gradnikov *(angl. Widgets)*. Z njimi razvijalci ne izboljšujejo samo obstoječih elementov uporabniškega vmesnika, temveč mu

dodajajo tudi nove. Njihova uporaba v obstoječih aplikacijah je brez pomembnega vpliva na ponovno preoblikovanje arhitekture aplikacije.

Tretja stopnja je raven implementacije z ogrodji (*angl. Frameworks)* ali orodjarnami (*angl. Toolkits*) na odjemalčevi strani. Ta kategorija je verjetno najbolj priljubljena med razvijalci. Tu se zavrže večino programske kode trenutne aplikacije in se jo na novo izvede s pomočjo določenega ogrodja ali orodjarne. Preoblikovanje v takem obsegu zahteva od razvijalca veliko več vloženega časa in dela. Vse to pa odtehtajo potencialne zmožnosti, ki jih lahko ponudi aplikacija zaradi velikega napredka na področju uporabniške izkušnje. To ponuja povečano odzivnost in uporabnost, kar vodi k večji produktivnosti in zadovoljstvu uporabnikov. Prav tako nudi razvijalcem boljšo doslednost pri pisanju programske kode in lažje upravljanje aplikacije kot celote.

Zadnja (četrta stopnja) raven implementacije je podobna prejšnji. Razlika je v tem, da je delovanje ogrodja na odjemalčevi strani tesno integrirano s komplementarnim delom strežniške strani. V nasprotju s prejšnjo ravnjo je ta odvisna od posebne strežniške platforme, kot so npr. *Java, .NET, PHP ...*



Slika 8: Razvrstitev orodjarn Ajax glede na raven uvedbe v aplikacijah

Primer aplikacije podjetja Yahoo, *Yahoo Mail*[10] – spletnega servisa za upravljanje z elektronsko pošto - nam ilustrira uporabo tega pristopa. V prvotni izvedbi je aplikacija temeljila le na prvi in drugi ravni uvedbe pristopa Ajax in bila je večkrat izboljšana s pomočjo različnih delov programske kode. Nedavno je bila v celoti prenovljena in predstavljena nova različica te aplikacije, ki v celoti temelji na tretji ravni uvedbe pristopa Ajax.

Kot vidimo, obstajajo različni načini, ki so na voljo pri uporabi pristopa Ajax, in število novih možnosti konstantno narašča z razvojem novih orodjarn in pristopov. Da bi kar največ pridobili z uporabo orodjarn Javascript, mora biti njena izbira dobro premišljena. Tako bomo

[10] http://overview.mail.yahoo.com/

v naslednjem razdelku podrobno spoznali in primerjali različne vidike delovanja dveh orodjarn Javascript tretje ravni.

4.3 Orodjarne za delo na odjemalčevi strani

Veliko podjetij ponuja na področju razvoja orodjarn Ajax svoje rešitve. *Google, Yahoo, Microsoft, Adobe* so le nekatera izmed največjih podjetij, ki si konkurirajo v tej tekmi. Nasproti temu imamo veliko razvijalcev, ki so združeni v manjših podjetjih ali v manjših skupinah. Plod njihovega razvoja so nekatere znane in uspešne orodjarne Ajax npr. *MooTools, jQuery, Dojo, Ext JS, YUI, Prototype* in *Scripaculus* ter druge. Njihovo število konstantno narašča, veliko pa je takih, ki so v svojem razvoju zaostale oziroma je bil njihov nadaljnji razvoj opuščen.

V naslednjem razdelku se bomo osredotočili le na dve bolj priljubljeni in razširjeni orodjarni Javascript. Vodilo pri njuni izbiri je, poleg približno enako dosežene stopnje razvitosti, tudi dejstvo, da sta na voljo v odprti kodi. To sta potrebna pogoja za smiselnost njune izbire. Njuna skupna značilnost je tudi v izvajanju na odjemalčevi strani v spletnem brskalniku in povezovanje s poljubno strežniško tehnologijo.

Potek primerjave ilustrira pomembnejše razlike pri izvedbi določenih funkcionalnosti obeh orodjarn Javascript. To odkriva njune različne pristope, iz česar se lahko naučimo, kdaj uporabiti posamezno orodjarno in kako se izogniti potencialnim pastem. Katero izbrati, je predvsem odvisno od tega, kaj želimo z njo doseči in v kolikšni meri hočemo dopolniti znanje jezika Javascript. Veliko zanimivih stvari ponujajo orodjarne Javascript in z njihovo uporabo se jih lahko naučimo.

Delovanje orodjarn Javascript temelji na jeziku Javascript, ki je bil med razvijalci uporabljen že v preteklosti, vendar se je ta z leti vedno bolj zmanjševal. Temu je botrovalo predvsem okolje za izvajanje - spletni brskalnik - in morda tudi dejstvo, da ni veljal za »pravi« programski jezik. V zadnjih letih je ponovno pridobil na veljavi (predvsem po zaslugi pristopa Ajax in aplikacij *RIA*). Uporaba jezika Javascript se je povečevala in postal je pomembnejši, tudi z uvajanjem nekaterih posebnih pristopov, npr. prototipno dedovanje, funkcijsko programiranje...

Danes je jezik Javascript posredno prisoten z uporabo orodjarn Javascript v številnih spletnih aplikacijah in je v povezavi z njimi eno bolj uporabljenih in priljubljenih orodij, ki omogoča izvedbo sodobnejših spletnih aplikacij v povezavi z odprto-kodnimi strežniškimi platformami ter je predvsem cenovno dostopnejša alternativa drugim konkurenčnim tehnologijam.

4.3.1 Primerjava orodjarn Mootools in Jquery – splošne razlike

Če smo natančni, bi orodjarno Mootools lahko označili za ogrodje. Poskuša izvesti jezik Javascript na način, kot bi ta »moral biti«. Če izhajamo iz definicije orodjarne Mootols, gre za pisanje objektno usmerjene, fleksibilne in kompaktne programske kode. Njen cilj je implementacija takega aplikacijskega programskega vmesnika, ki daje občutek uporabe jezika

Javascript tako, da krepi celoten spekter funkcionalnosti jezika (ne osredotoča se samo na eno stvar, npr. delo s tehniko *DOM*).

Orodjarna Jquery se osredotoča predvsem na zagotavljanje lažjega in prijaznejšega načina dela s tehniko *DOM*. Vsebuje celotno zbirko metod, ki to omogočajo. Če izhajamo iz definicije orodjarne Jquery, kjer je drugačen princip razmišljanja pri dodajanju obogatenih funkcionalnosti spletnim aplikacijam. Za razliko od orodjarne Mootools se tu ne porablja časa za napredne koncepte jezika Javascript, ampak je glavna osredotočenost ustvarjalcev orodjarne na zagotavljanju enostavnejše manipulacije nad elementi *DOM*.

Učljivost in podpora skupnosti

Orodjarna Jquery je v glavnem lažje učljiva in najbolj uporabna v primerih, ko želimo kakšno stvar hitro spraviti v delovanje, brez pretiranega poznavanja jezika Javascript. V tem primeru je orodjarna Jquery primernejša v primerjavi z orodjarno Mootools. Problem ni v tem, da z orodjarno Mootools ne bi mogli opraviti istih opravil, vendar je z njo, ob manjšem poznavanju jezika Javascript, to težje doseči. V prid orodjarni Jquery je tudi veliko spletnih virov (več kot jih obstaja za orodjarno Mootools), ki so nam lahko v pomoč pri učenju.

Pri primerjavi obeh skupnosti je zaznati večjo aktivnost in večje število uporabnikov orodjarne Jquery. Razlog za to je predvsem v njeni hitri učljivosti (zaradi njene enostavnosti) in v veliko bolj intenzivnem promoviranju. Večja priljubljenost orodjarne Jquery se pokaže tudi pri meritvah števila prenosov orodjarne, prodanih knjig in izvedenih iskanj v spletnih iskalnikih.

Nivo implementacije funkcionalnosti

Orodjarno Jquery odlikuje izrazit sistem za opisovanje obnašanja v spletni aplikaciji in včasih ta ne daje občutka »pravega« programiranja. Pri tem se v večini osredotoča na tehniko *DOM* in implementiranje funkcije za izvajanje s tem povezanih nalog (spreminjanje lastnosti *CSS*, animiranje elementov, prejemanje vsebin preko Ajax-a, ipd). Poleg tega zagotavlja še nekatere druge metode (za urejanje nizov, premikanje po tabelah, itd.), ki niso povezane s tehniko *DOM*, vendar so splošno koristne. Teh je manj kot pri orodjarni Mootools. Veliko dodatnih metod pravzaprav orodjarna Jquery niti ne potrebuje, saj je večina nalog usmerjena v uporabo tehnike *DOM*, spreminjanju elementov (dodajanje elementov *HTML*, spreminjanje njihovih slogov, dodajanje poslušalcev, dogodkov, itd). Vse ostale funkcionalnosti, ki jih orodjarna Jquery ne pokriva, je potrebno opraviti s pomočjo običajnega jezika Javascript.

Če pomislimo na celotno področje delovanja jezika Javascript, vidimo, da je orodjarna Jquery precej ozko usmerjena, saj večinoma ne podpira funkcionalnosti zunaj uporabe tehnike *DOM*. Torej, njen glavni cilj je: biti programski sistem pretežno za delo s to tehniko. To je tudi eden od razlogov njene hitrejše učljivosti, hkrati s tem omejuje načine, s katerimi bi lahko olajšala delo jezika Javascript. Ne obravnava koncepta dedovanja, razredov, podrazredov, vmesnikov, obravnava pa koristne metode za prirojene osnovne podatkovne tipe, funkcije, tabele jezika Javascript. Torej, če povzamemo, bi lahko zaključili s tem, da je orodjarna Jquery najbolj uporabna pri delu s tehniko *DOM*. Za ostalo rabo, ki obsega funkcionalnosti izven tega področja, je bolje uporabiti alternativno orodjarno.

Orodjarna Mootools se v tem pogledu precej razlikuje. Namesto, da se osredotoča izključno na delo s tehniko *DOM*, je njeno delovanje usmerjeno v celoten spekter jezika Javascript. Poskuša zajeti vse njegove aspekte, tudi zahtevnejše. V začetku zahteva več vloženega truda in predznanja zaradi višjega nivoja implementacije posameznih funkcionalnosti in posledično večje zahtevnosti. Na daljši rok se njena uporaba izkaže za primernejšo, če jo primerjamo z orodjarno Jquery, in sicer pri delu s kompleksnejšimi spletnimi aplikacijami.

Moč in izraznost

Moč orodjarne Mootools se pokaže pri izdelavi bolj stabilnega in skladnega aplikacijskega programskega vmesnika, kar omogoča, da je razvoj z jezikom Javascript v celoti manj frustrirajoč. Orodjarno Mootools lahko razumemo kot razširitev jezika Javascript. To izraža s ponujanjem bolj uporabnih in dostopnih funkcionalnosti (prototipnega dedovanja, samo sklicevanja, povezovanja), kot jih poznamo pri jeziku Javascript. Pomemben del jedra te knjižnice je namenjen povečanemu obsegu delovanja osnovnih objektov oziroma prototipov, kot so: funkcije, nizi, tabele, števila, elementi. Poleg tega ponuja funkcijo, imenovano *Razred*. Z njo omogoča lažjo uporabo modela prototipnega dedovanja.

Vsi omenjeni koncepti niso edinstveni v orodjarni Mootools (ponujajo jih tudi druge orodjarne), vendar nekatere izmed njih orodjarna Jquery ne zagotavlja v taki meri. Jedro orodjarne Jquery ne ponuja sistema dedovanja, implementacije funkcionalnosti enega razreda, ki so lahko uporabne znotraj drugih razredov (sistem vmesnikov), prav tako ne ponuja izboljšav prvotnih objektov (funkcij, nizov, itd.) jezika Javascript. Verjetno bi orodjarna Jquery lahko ponudila omenjene koncepte, vendar so se snovalci orodjarne odločili omejiti področje uporabe.

Selektorji in veriženje

Orodjarna Jquery zagotavlja vsestranski in zmogljiv nabor selektorjev za identifikacijo elementov znotraj dokumenta *HTML*. S pomočjo strnjene sintakse in funkcije *$()*[11] lahko določimo selektorje s sintakso po standardu *CSS2, CSS3 in* tudi izvedbo nekaterih lastnih selektorjev. Ti pridejo v poštev takrat, ko želimo izbrati elemente, ki temeljijo na značilnostih in te niso predvidene v specifikaciji *CSS*. Na primer, če hočemo omejiti množico elementov na vsa potrditvena polja, ki jih je uporabnik označil, lahko to zapišemo s sintakso *$(':checkbox:checked')*. S takim načinom določevanja lahko enostavno in natančno opredelimo želeno množico elementov, s katerimi hočemo manipulirati. V ta namen nam orodjarna Jquery zagotavlja skupek metod, ki jih lahko uporabimo kar z veriženjem ukazov na objektu *$()*.

Delovanje in optimizacija pogona, ki služi izbiranju elementov *DOM*, je dokaj ključnega pomena, saj temelji velik del izvedenih operacij vsake orodjarne prav na omenjenih

[11] Funkcija *$()* (drugo ime za funkcijo *Jquery()*) vrne poseben objekt Javascript, ki vsebuje niz elementov *DOM*, ki se ujemajo z določenim selektorjem. Ta objekt vsebuje veliko uporabnih in vnaprej določenih metod, ki delujejo na pridobljeni množici elementov.

poizvedbah s pomočjo selektorjev. S tem se lahko precej pospeši oziroma upočasni delovanje orodjarne, če pogon ni najbolje zasnovan.

Orodjarna Jquery je v zadnji različici prešla na novo različico Pogon za izvajanje poizvedb s selektorji, ki je precej pospešil delovanje v primerjavi s predhodno različico (za ~40%). Mogoča je tudi njegova razširitev z uporabo vtičnikov, tako da to še poveča izraznost določevanja sintakse pri zbiranju elementov *DOM*. Deluje neodvisno od ostalih delov orodjarne z namenom, da ga lahko uporabljajo tudi druge konkurenčne orodjarne. Primerjava zmogljivosti tega pogona s pogonom za izbiro elementov orodjarne Mootools pokaže, da so skupni časi dostopov z najbolj uporabljenimi selektorji[12] za različne spletne brskalnike v orodjarni Jquery hitrejši kot v orodjarni Mootools (Tabela 4.1)[13]. Prav tako je to eden hitrejših, če ne najhitrejši pogon, med vsemi orodjarnami na trgu.

Orodjarna / Spletni brskalnik	jQuery 1.2.6	jQuery 1.3	MooTools 1.2.1
Firefox 3	184	111	240
Firefox 3.5	113	34	135
Safari 3.2	71	15	76
Opera 9.6	107	75	132
IE 6	854	640	1611
IE 7	210	181	490
Chrome	30	13	118

Tabela 4.1: Skupni časi dostopa z uporabo selektorjev (časi so milisekundah)

Orodjarna Mootools prav tako vsebuje funkciji, s pomočjo katerih izbiramo elemente *DOM*, nad katerimi želimo manipulirati. Funkcija *$()* omogoča izbor elementov glede na njihov identifikator, medtem ko imamo še funkcijo *$$()*, katere namen omogoča izbor množice elementov s pomočjo selektorjev *CSS*. Torej imamo, za razliko od orodjarne Jquery, dve ločeni funkciji, ki omogočata izbor elementov. Število lastnih selektorjev, ki jih je mogoče uporabiti v orodjarni Mootools, je manjše kot v orodjarni Jquery. Poleg izbora elementov je naloga funkcij *$()* in *$$()* tudi njihova inicializacij z dodajanjem pripadajočih metod, ki jih lahko izvedemo na njih z uporabo orodjarne Mootools. Tako je vsak pridobljen element avtomatično razširjen z metodami objekta *Element*[14] (objekt Javascript, ki se nanaša na oznake *HTML*). Pri orodjarni Mootools so te metode bolje izvedene, njihova uporaba ni omejena le na elemente, razširjene s pomočjo funkcije *$()* in *$$()*, ampak tudi na ostale. V orodjarni Jquery ni tako, saj so metode pri izbranih elementih *DOM* na voljo le v primeru, če so ti razširjeni s pomočjo funkcije *Jquery()* (ali krajše *$()*).

[12] http://ejohn.org/files/selectors.html
[13] http://docs.jquery.com/Release:jQuery_1.3#Performance
[14] http://mootools.net/docs/core/Element/Element

Vzorec veriženja ja prisoten v orodjarni Mootools in v Jquery. Obe ga uporabljata kot del svoje sintakse, kjer se s pomočjo veriženja ene metode na drugo lahko izvede veliko aktivnosti v eni vrstici programske logike. Tak način izvedbe omogoča, poleg pisanja jedrnatih operacij, tudi izboljšano zmogljivost. Ob tem ni potrebno vsakič znova pridobivati elementov iz strukture *DOM*, na katerih želimo izvesti več operacij hkrati. Primer programske kode (19) prikazuje način uporabe tega vzorca v orodjarni Mootools.

```
window.addEvent('domready', function() {
    var selektor = $('selektor');     //selektor-poljuben element
                                      //dokumenta HTML
    faq.getElements('dd').hide();
    faq.getElements('dt').addEvent('click', function() {
            this.getNext().slide('toggle');
    });
});
```
(9)

Zgoraj prikazana programska logika je enaka v primeru (10), kjer je zapisana s pomočjo orodjarne Jquery.

```
$(document).ready(function() {
    $('#selektor').find('dd').hide().end().find('dt').click(function() {
          $(this).next().slideToggle();
    });
});
```
(10)

Spoznamo lahko, da uporablja orodjarna Mootools bolj razvlečeno, ampak zaradi tega tudi jasnejšo sintakso, orodjarna Jquery veliko bolj strnjeno. Slednji način pisanja se vedno ne izkaže za najboljšega, saj lahko povzroča težjo berljivost in razumljivost zapisane programske logike. To daje pogosto občutek negotovosti, kaj je namen določenega sklopa programske logike (ob dobrem poznavanju dokumentacije je jasno, kaj posamezna metoda naredi, vendar nekatera njihova poimenovanja v orodjarni Jquery pogostokrat ne dajejo tega občutka). To lahko predstavlja težavo, ko mora razvijalec uporabiti programsko logiko drugega razvijalca. Podpora s komentarji je v takih primerih potrebna, drugače je bolje uporabiti manj strnjen način pisanja, za kar se zdi primernejša orodjarna Mootols.

Vzorec dedovanja, razširjanja in ponovna raba programske logike

V orodjarni Mootools se model dedovanja nekoliko razlikuje od mnogih drugih programskih jezikov. Tu se dedovanje vrši s pomočjo funkcije *Razred*[15]. To ime sicer spominja na sistem tradicionalnega dedovanja, kot ga poznamo npr. iz jezika Java, vendar z njim nima veliko skupnega. Tako poimenovanje se uporablja predvsem z namenom ustreznega opisa tega, čemur je funkcija namenjena. Dedovanje v orodjarni Mootools je posebno prototipno, v katerem dedujejo objekti neposredno od drugih objektov. To se vrši preko posebne skrite povezave, ki jo ima vsak objekt do svojega nadrejenega objekta.

[15] Ko se sklicujemo na razrede v povezavi z orodjarno Mootools, mislimo na funkcije, ki vračajo objekte (primerke objektov).

Povezavo določa lastnost z imenom *prototype.* Če vzamemo za primer tabelo, katere primerek ustvarimo, bo ta najprej dedovala od prototipa tabele, na katerega se lahko sklicujemo z *array.prototype.* Vse opravljene spremembe na tem prototipu se takoj odrazijo na vseh primerkih tabel. Če bi tako želeli dodati tabeli metodo za seštevanje, bi to lahko storili z:

```
array.prototype.sestevek = function () {
     //telo funkciije
}
```
(11)

S tem smo ustvarili lastnost prototipa tabele, ki ga lahko uporabimo v vseh drugih primerkih tabel. V primeru klica neobstoječe metode določenega objekta se njena prisotnost rekurzivno preveri v celotni verigi dedovanih objektov, to je do najvišjega prototipa objekta *Object.prototype.*

Pri tem načinu dedovanja se ne določa nobenega vzorca v obliki razreda, niti se jih ne ustvarja, temveč, kot že rečeno, gre za uporabo funkcije z imenom *Razred*. Če se sklicujemo nanjo (12), mislimo na funkcijo, ki kot argument sprejme podatkovno strukturo objekt (ta postane prototip vsakega primerka tega razreda) .

```
var nov_razred = new Class({
     initialize: function(arg_1, arg_2) {
          this.arg_1 = arg_1;
          this.arg_2 = arg_2;
   },

   ime_lastnosti_1: 1,
   ime_lastnosti_2: 2,
   ime_metode: function() {
           //telo metode
   }
});

//primerek razreda
var primerek = new nov_razred(arg_1, arg_2);
```
(12)

Za razliko od orodjarne Jquery lahko pri orodjarni Mootools enostavno razširimo delovanje v podrazrede (13) in jim dodamo nove funkcionalnosti. V tem primeru se prepiše osnovna metoda (konstruktor) z imenom *initialize* v nadrejenem razredu, na katero se lahko še vedno sklicujemo s klicem *this.parent* v metodi *initialize* podrazreda. S tem imamo možnost nadziranja, kdaj se določena programska logika izvrši – to je pred ali po klicu nadrejenega razreda, imamo tudi možnost poljubnega dostopanja do metod in lastnosti ter določanja novih vrednosti.

```
var pod_razred = new Class({
     Extends: nov_razred,
          Initialize: function(param, param_1, param_2) {
               this.param = param;
               this.parent(param_1, param_2);
          },
          Ime_lastnosti_3: 3,
```

```
            Ime_metode_1: function() {
                 //telo metode
            }
 });
                                                                    (13)
```

Poleg tega imamo v orodjarni Mootools še en vzorec, ki omogoča, za razliko od razširjanja razreda v podrazrede, uporabo več razredov znotraj enega. To so tako imenovane mešanice razredov (14), kjer je osnovni razred prežet z lastnostmi vseh vanj vključenih razredov. Na ta način bi lahko na primer določili razred samih pomožnih funkcij (angl. *helper class*), ki bi ga po potrebi vključevali v osnovne razrede, v katerih bi se pokazala potreba po uporabi pomožnih funkcij.

```
var pod_razred = new Class({
     Extends: nov_razred,
     //v tabeli lahko določimo več razredov, katerih lastnosti želimo
     //vključiti
     Implements: [razred_1, razred_2, razred_3]
     initialize: function(arg_1, arg_2) {
             this.parent(name, age);
     }
});
                                                                    (14)
```

Ob takem načinu izvedbe morda porabimo več časa za pisanje in oblikovanje vseh razredov, vendar se s tem izognemo nepotrebnemu ponavljanju programske logike in ohranimo podrobno stopnjo nadzora nad tem, kdaj so posamezne metode poklicane in kako so medsebojno povezane. Končni rezultat lahko obsega nekaj več vrstic, kot bi jih porabili v orodjarni Jquery, vendar je tu prednost v fleksibilnejši in bolj berljivi programski logiki. Fleksibilnost je lastnost, ki je dobrodošla predvsem pri integraciji novih delov programske logike s starimi, v tem je orodjarna Mootools uspešnejša od orodjarne Jquery.

Orodjarna Mootools ne ponuja tako velikega števila gradnikov in ti tudi niso uradno vodeni v posebni zbirki, kot to velja za orodjarno Jquery. Pomagamo si z gradniki posameznih razvijalcev, vendar nimamo zagotovila za njihovo kompatibilnost, in sicer v primeru izdaje nove različice orodjarne. Večina tako pridobljenih gradnikov lahko reši naš problem, v nasprotnem primeru jih lahko še vedno razširimo in dodamo lastne funkcionalnosti, ki jih potrebujemo. To bi bil lahko eden glavnih razlogov manjšega števila gradnikov, ki ji ponuja orodjarna Mootools.

Orodjarna Jquery vsebuje manjši nivo že omenjenih funkcionalnosti orodjarne Motootols. Razširjanje in ponovno rabo programske logike poskuša opraviti s pomočjo sistema vtičnikov, ki so vedno vezani le na elemente *DOM*. Jedro knjižnice ne podpira sistema dedovanja tako, kot ga poznamo pri orodjarni Mootools. Ne ponuja sistema razširjanja razredov v podrazrede in ne tako imenovanih mešanic razredov. Razširitve in izboljšave je mogoče izvesti le na primeru lastnega objekta *Jquery, to je* v dveh oblikah:

– kot splošno koristne (globalne) funkcije. To so pravzaprav metode objekta *Jquery*. Praktično gledano, jih uporabljamo kot funkcije znotraj njegovega imenskega

prostora in niso namenjene delovanju na elementih *DOM*. Nudijo nam bližnjice za izvedbo nalog, ki jih pogosto izvajamo na primer, uporaba funkcij *$.each()*, *$.trim()*…;

– kot metode za operiranje nad dobljeno množico elementov s pomočjo funkcije *$()*. Tu lahko dodamo lastne metode kot lastnosti objekta, imenovanega *Jquery.fn*[16].

Kot vidimo, dodajanje globalnih funkcij zahteva razširjanje objekta *Jquery* z novimi metodami, med tem ko dodajanje funkcij za delo nad množico elementov *DOM* zahteva razširitev prototipa objekta *Jquery.fn.* Prvo obliko razširitve lahko zapišemo z uporabo vtičnika v obliki (15), ki objema za ta vtičnik novo ustvarjene globalne funkcije v objekt. To onemogoča konflikte z drugimi imeni funkcij in spremenljivk, uporabljenih znotraj imenskega prostora objekta *Jquery*.

```
jQuery.mojVticnik = {
     funkcijaEna: function() {
          //telo funkcije
     },
     funkcijaDve: function(parameter) {
          //telo funkcije
     }
};
```
(15)

V drugi obliki razširitve, kjer gre za dodajanje lastnih metod za operiranje nad izbranimi elementi *DOM*, se uporablja splošen vzorec (16).

```
(function($){
     $.fn.imeFunkcije = function() {
          //telo funkcije
     }
})(jQuery);
```
(16)

Definicija znotraj zunanje funkcije zagotavlja, da lahko znak *$* uporabimo kot drugo ime za objekt *Jquery*, ob morebitni uporabi še katere druge orodjarne Javascritp, in zagotovimo, da ne pride do konflikta.

Druga oblika spodbuja tudi pisanje ponovno uporabljive programske logike v obliki vtičnikov. Njihova izvedba ni tako kompleksna, kot je uporaba razredov v orodjarni Mootools, saj so to posamezne funkcije, pogosto smo priča vzorcu razširjanja (17).

```
jQuery.fn.razsirjen_vticnik = function(options) {
    var obj = jQuery.extend({
            param1: vrednost1;
            param2: vrednost2
    }, options);

    //tu dodamo novo logiko v razširjen vtičnik
    this.vticnik; //klic vtičnika, ki ga razširjamo
```

[16] *Jquery.fn* je skrajšano ime za *Jquery.prototype*, s pomočjo katerega lahko razširimo delovanje ustvarjenih primerkov objekta *Jquery*.

```
});
```
(17)

Z njim izvedemo vtičnik z dodajanjem nove programske logike, razširjanjem objektov s pomočjo funkcije *extend()* in klicem vtičnika, katerega funkcionalnosti želimo razširiti. Pogosto se zgodi, da se pri takem načinu razširjanja ne moremo izogniti podvajanju določenih delov programske logike (npr. uporaba identičnih selektorjev za izbor elementov *DOM* v vtičniku, ki ga razširjamo, in v novo dodani programski logiki), kar je zelo potratno.

V orodjarni Jquery je mogoča izdelava tudi zahtevnejših vtičnikov z metodami in stanji, vendar je tak vzorec podprt v posebno ločenem sistemu vtičnikov oziroma zbirki gradnikov, poimenovani *Jquery UI*, ki pa ne uporablja enakega mehanizma delovanja vtičnikov, kot je bil predstavljen v nalogi. Sicer je v tem primeru orodjarna Jquery, če jo primerjamo z orodjarno Mootools, v prednosti, saj zagotavlja uradno vodeno zbirko gradnikov, ki jo orodjarna Mootools ne ponuja. Vendar je to dodatek k osnovni orodjarni Jquery in ne del njene osnovne izdaje. Če se dosledno osredotočimo na primerjavo samo tistih funkcionalnosti obeh orodjarn, ki jih ponujata v svojem jedru, tega sistema (*Jquery UI*) ne moremo upoštevati v tej primerjavi.

Ker se pogosto med razvojem aplikacij srečamo s programsko logiko, podvojeno na različnih mestih, je dobra praksa uporaba prikazanih rešitev v tem razdelku, ki nam jih ponujata orodjarni. To zagotavlja zmanjšanje količine programske logike in optimizira izvajanje. Smiselnost njihove uporabe se pokaže tudi iz naslednjih dveh razlogov:

- pri izdajanju novih različic orodjarn in posledično sprememb delovanja njenih funkcij je potrebno večkratno spreminjanje programske logike. S tem ohranjamo večjo konsistentnost in omogočimo lažjo in preglednejšo odpravo napak, dodajanje ali spreminjanje funkcionalnosti;
- zmanjšanje količine programske logike s ponovno rabo vedno istih metod in programskih sklopov. To pomeni pospešitev delovanja in hitrejši začetni prenos aplikacije.

Zaključne ugotovitve primerjave

Kot smo videli, se orodjarna Jquery osredotoča na izvedbo enostavnega in hitrega programiranja, predvsem na tehniko *DOM*. Orodjarna Mootools je bolj usmerjena na razširitve, dedovanje, ponovno rabo, vzdrževanje in čitljivost programske logike. Ob primerjavi tega vidimo, da je z orodjarno Jquery lažje začeti in tudi hitreje pridemo do prvih rezultatov. Izkaže se, da sta možnosti ponovne rabe in vzdrževanja programske logike, zapisane s pomočjo orodjarne Jquery, težje izvedljivi. Na drugi strani je pri uporabi orodjarne Motools potrebno več predhodnega znanja jezika Javacript, potrebno je vložiti tudi več časa v učenje in spoznavanje orodjarne ter napisati več programske logike, preden pridemo do prvih rezultatov. Posledica tega je lažje izvajanje vzdrževanja in ponovne rabe programske logike.

Obe orodjarni ne vsebujeta vseh mogočih funkcionalnosti, ki si jih lahko zamislimo, ampak ohranjata čim manjšo velikost svojih jeder in zagotavljata le najnujnejše. Namesto tega

nam dajeta možnost izvedbe lastnih funkcionalnosti, s katerimi lahko izvedemo vse tisto, kar si zamislimo. To je moč jezika Javascript in orodjarn Javascript.

Orodjarna Mootools uporablja bolj celosten pristop, kar upočasnjuje hitrost napredka pri njenem učenju. S takim pristopom mislimo na področje delovanja, ki ni ozko usmerjeno, ampak zajema širše področje delovanja jezika Javascript. Orodjarna Jquery je ožje usmerjena, osredotočena je le na področje *DOM*. Ko gre za izvrševanje nalog, ki niso vezane na to področje, je njena uporaba onemogočena oziroma smo prisiljeni uporabiti običajni jezik Javascript.

Katera orodjarna je primernejša, ni odvisno samo od omenjenih prednosti in slabosti, ampak je potrebno poudariti razliko v filozofiji obeh. Če želimo hitro videti rezultate opravljenega dela določene stvari s čim manj vloženega časa, bomo zagotovo izbrali orodjarno Jquery. V nasprotnem primeru, če želimo delati na daljši rok, bomo izbrali orodjarno Mootools.

4.3.1 Orodjarni Mootools in Jquery – izvedba tehnike Ajax

Eden od vzrokov nastanka in popularizacije orodjarn Javascript je zagotovo v tehniki Ajax, ki je najbolj povezana s frazo *»Web 2.0«* in vključena že v vsako orodjarno Javascript.

Način izvedbe Ajax-a s pomočjo navadnega jezika Javascript smo že spoznali v nalogi, ko smo govorili o tehnični izvedbi zahtevka Ajax s pomočjo navadnega objekta *XHR*. Ker že vemo za očitne težave v delovanju tega objekta, je to pravi trenutek, da lahko uporabimo orodjarne Javascript. Olajšajo nam delo s tehniko Ajax, kar naredi njeno uporabo prijetnejšo. V prvi vrsti gre za poenostavitev posameznih funkcionalnosti, ki smo jih pred tem morali izvesti z navadnim jezikom Javascript. Odpravljajo tudi večino nekonsistentnosti, za katere smo morali poskrbeti sami. Med drugim, ni nam več potrebno skrbeti za različne izvedbe objekta *XHR* v različnih spletnih brskalnikih. V tem razdelku bomo spoznali funkcije obeh orodjarn, ki jih uporabljata za izvedbo tehnike Ajax. Velikokrat so ravno te vzrok za uporabo orodjarn Javascript.

Izvedba tehnike Ajax – orodjarna Mootools

Orodjarna Mootools uporablja za izvedbo tehnike Ajax tri razrede, ki olajšajo delo z njo. To je razred *Request* in dve njegovi razširitvi, podrazreda *Request.HTML* in *Request.JSON* [14].

Uporaba razredov ni vedno nujna in tako tudi njihovo podrobno poznavanje ne. Namesto tega lahko uporabljamo različne bližnjice, ki jih v ta namen zagotavlja orodjarna Mootools. Na primer: na elementih *DOM,* ki omogočajo nalaganje vsebine s pomočjo Ajax-a. Primer takega delovanja sta metodi *send()* in *load().* Njuna uporaba in razumevanje sta enostavnejši kot že omenjeni razredi. Prva metoda omogoča enostavno pošiljanje informacij na podlagi vsebine kateregakoli elementa *DOM*, ki vsebuje elemente *HTML*, tipa *input* (običajno v obrazcu). Primer (18) prikazuje način pošiljanja obrazca s pomočjo Ajax-a. Tu so vsi parametri in vrednosti vnosnih polj uporabljeni na enak način kot pri navadnem pošiljanju spletnega obrazca. Razlika je le v uporabi Ajax-a, ki pošlje podatke asinhrono.

```
$('mojObrazec').send(naslov_url);
```

```
//ali
$('mojObrazec').set('send', {
 onSuccess: function(response){
      //telo funkcije
}
}).addEvent('submit', function(event){
      this.send(); //uporabi se naslov URL obrazca parametra »action«
});
```
(18)

Druga metoda (19) omogoča hitro posodobitev vsebine elementa *DOM*, nad katerim jo izvršimo. To storimo z minimalno količino programske logike.

```
//uporaba
$('mojElement').load(url, [podatki]);

//primer uporabe
$('mojElement').load('/stran.html');
```
(19)

Če hočemo zagotoviti celovito delo s pristopom Ajax, je običajno potrebna že omenjena uporaba razredov. Le ti nam lahko zagotavljajo popolno kontrolo nad njegovim izvajanjem. Razred *Request* je namenjen osnovni interakciji *zahtevek/odgovor*, ko je potrebno asinhrono pošiljanje in/ali sprejemanje podatkov na strežnik ali z njega, ter služi tudi kot osnova ostalima dvema podrazredoma. Z njegovo uporabo pridobimo veliko dodatnih funkcionalnosti in številne nove možnosti uporabe z določanjem raznovrstnih lastnosti. Nov primerek razreda *Request* določimo tako:

```
//uporaba
new Request (nastavitve)

//primer uporabe
var mojZahtevek = new Request({
      url: '/requestHandler.php',
      method: 'get',
      onSuccess: function(responseText, responseXML) {
            //telo funkcije
      }
});
```
(20)

V primeru (20) ne prikazujemo vseh možnih lastnosti, ki bi jih lahko določili znotraj razreda *Request*. Vse lastnosti so navedene v dokumentaciji[17] orodjarne Mootools. Torej, primer (21) prikazuje uporabo opcijskega parametra *data*, s katerim lahko določimo pošiljanje prevzetih podatkov kar znotraj razreda.

```
var mojZahtevek = new Request({
      url: '/requestHandler.php',
      method: 'get',
      data: {
            id: '1000'
      }
```

[17] http://mootools.net/docs/core/Request/Request

```
        onComplete: function(responseText, responseXML) {
            //telo funkcije
        }
    });
```
(21)

Veliko je še dodatnih lastnosti, ki jih je mogoče koristno uporabiti. Ena iz med njih je zagotovo uporaba dogodkov, ki jih lahko določimo objektu razreda *Request*. Najpomembnejši je dogodek *onComplete*, znotraj katerega pridobimo želene podatke v primeru uspešno vrnjenega odgovora (dogodek *onSuccess*) ali pa ustrezno napako (npr. 404) v primeru neuspešnega odgovora (dogodek *onFailure*).

V obeh primerih (20, 21) ustvarimo le primerka razreda *Request*, nobenih podatkov se ne pošilja. Šele ob izvršenem klicu metode *send()* na tem objektu, se zahtevek dejansko pošlje na strežnik. Torej moramo v obeh primerih podatke poslati, kar storimo na tako:

```
//uporaba
mojZahtevek.send();
mojZahtevek.send( {ime:'vrednost'} ); //argument objekt
mojZahtevek.send( 'ime=vrednost&ime1=vrednost1' ) //argument niz znakov

//primer uporabe
//pošiljanje privzetih podatkov
mojZahtevek.send();
//prepis obstoječih podatkov
mojZahtevek.send({id: '1001'});
```
(22)

Primer uporabe (22) prikazuje pošiljanje zahtevka iz primera (21) s pomočjo metode *send()*. To lahko storimo na dva načina, z uporabo opcijskega argumenta ali brez njega. Če opcijski argument določimo, imamo možnost večkratne uporabe istega primerka objekta za pošiljanje različnih podatkov.

Pogost vzorec za prikaz podatkov na zahtevo s pomočjo Ajax-a je z uporabo razreda *Request.HTML*. To je razširitev razreda *Request*, ki avtomatizira posodabljanje elementov *DOM* z odzivom na zahtevo Ajax. Je posebej izdelan za prejem podatkov v obliki *HTML*. Poleg vseh nastavitev, ki jih podeduje od razreda *Request*, vsebuje še druge dodatne nastavitve. Primer (23) prikazuje uporabo lastnosti *update*, s katero določimo element *DOM*, ki se ob prejetem odgovoru samodejno posodobi s prejeto vsebino.

```
//uporaba
new Request.HTML(nastavitve);

//primer uporabe
new Request.HTML({
    url: '/profil_uporabnika.php',
    data: {
        id: '1000'
    },
    update: $('profil_id');
}).send();
```
(23)

Razred *Request.JSON* je še ena razširitev razreda *Request*, ki jo vsebuje orodjarna Mootools in avtomatizira pošiljanje in sprejemanje podatkov v obliki *JSON*. Uporablja se predvsem kot alternativa formata *XML* za prenos podatkov med odjemalcem in strežnikom. Zaradi svoje zgoščene strukture ne vsebuje odvečnih elementov, kar pohitri prenos podatkov. Pomembna prednost pri uporabi formata *JSON*, v primerjavi s formatom *XML*, se izkaže v tem, da ga razčlenjevalnik Javascript razume. Na ta način je veliko lažje razčlenjevanje podatkov, kot smo jih vajeni z uporabo formata *XML*. Za to uporabljamo enostavno notacijo s pomočjo operatorja pika (24).

```
//uporaba
new Request.JSON(options);

//primer uporabe
var jsonRequest = new Request.JSON({
     url: " /script.php",
     onComplete: function(oseba){
          alert(oseba.starost);
          alert(oseba.visina);
          alert(oseba.teza);
     }
}).get({'ime': 'Janez', 'priimek': 'Novak'});
```
(24)

Izvedba tehnike Ajax – orodjarna Jquery

Orodjarna Jquery vsebuje več funkcij, s pomočjo katerih lahko izvršimo operacije v povezavi z Ajax-om. Tako tiste zahtevnejše (npr. *$.ajax()*), ki nudijo popolni nadzor nad izvedbo, kot ostale hitre (*npr. load(), $.get()*), vendar nepopolne rešitve [04].

Od teh je najbolj priljubljena funkcija *load()*, ki omogoča enostaven prikaz in posodobitev pridobljenih podatkov znotraj elementov *DOM* (npr. elementa <*div*>). Klic lahko izvedemo z minimalno količino programske kode, brez dodatnih nastavitev na naslednji način:

```
    //uporaba
    load(url, parametri, povratni_klic)

    //primer uporabe 1
    $(“div”).load(“datoteka.txt”);

    //primer uporabe 2
    $(document).ready(function( ) {
          $(“div”).load(“datoteka.txt”, povratni_klic);
    });

    function povratni_klic( ) {
          $(“#div_1”).text(“Sporočilo o statusu zahtevka.”);
    }

    //primer uporabe 3
    $(document).ready(function( ){
          $(“div”).load(“skripta.php”, {data: 1});
```

```
});
                                                                    (25)
```

Funkcija v primeru (25) uporablja tri argumente: *url* je naslov spletnega vira, iz katerega želimo pridobiti podatke, *parametri* je objekt Javascript, katerega lastnosti vsebujejo vrednosti, ki jih želimo poslati na strežnik, in *povratni_klic* je povratno-klicna funkcija, ki se pokliče ob zaključenem zahtevku Ajax. Če parametri določijo argumentu objekt Javascript s pripadajočimi vrednostmi, se ta pošlje s pomočjo metode *POST*. V nasprotnem primeru se uporabi privzeta metoda *GET*. Povratno-klicno funkcijo lahko določimo, ni pa nujna. Pokliče se tudi ob neuspešno izvršenem zahtevku. V njej običajno navedemo element *DOM*, npr. * *, v katerem se prikaze določeno sporočilo. Torej, sporočilo o napaki, če je izvedba zahtevka neuspešna.

Poleg funkcije *load()* ponuja orodjarna Jquery uporabo še dveh funkcij za eksplicitno pošiljanje zahtevkov po metodah *POST (*$*.post())* in *GET (*$*.get())*. Če hočemo več nadzora nad izvedbo zahtevkov in pridobljenimi podatki (preden ti samodejno posodobijo element *DOM*), lahko uporabimo ti dve metodi (26). Podobni sta prej omenjeni funkciji *load()* z razliko, da izvedeta povratno-klicno funkcijo le v primeru uspešne pridobitve podatkov.

Imata tudi dodaten argument tip, s katerim določimo obliko vrnjenih podatkov (npr. *XML*, *JSON*, *HTML*, navaden tekst).

```
//izvedba zahtevka z metodo POST
$.get(url, podatki, povratni_klic, tip)

//izvedba zahtevka z metodo GET
$.get(url, podatki, povratni_klic, tip)
                                                                    (26)
```

Orodjarna Jquery prav tako ponuja pošiljanje spletnega obrazca s pomočjo Ajax-a, vendar ta lastnost tu ni izvedena kot pri orodjarni Mootools (razširitev elementa *DOM*), ampak v ta namen uporabimo kar metodo *load()*. Za njen argument *podatki* uporabimo vrnjene podatke v obliki *JSON*, ki jih pridobimo s pomočjo funkcije *serializeArray()*, izvedene na želenem spletnem obrazcu.

Funkcije in ukazi, ki smo jih spoznali, so uporabni v številnih primerih, vendar včasih želimo oziroma potrebujemo večjo stopnjo nadzora z določanjem različnih lastnosti, ki nam omogočajo večjo svobodo pri izvedbi zahtevkov, vendar s pomočjo tehnike Ajax. Orodjarna Jquery zagotavlja splošno koristno funkcijo (27) za izdelavo takih zahtevkov, imenovano *$.ajax()*.

```
$.ajax( nastavitve )
//ali
$.ajax({
 url: "script.php",
 global: false,
 type: "POST",
 data: ({id : 1})
 dataType: "html",
 success: function(){
```

```
        //telo funkcije
    }
});
                                                                    (27)
```

Na mestu parametra nastavitve v primeru (27) lahko uporabimo objekt, njegove lastnosti določimo z različnimi parametri (bolj pogosto uporabljeni so: *url, tip, podatki, tip_podatkov* itd.). Z uravnavanjem teh nastavitev se lahko natančno določi izvedbo zahtevkov. Celoten seznam parametrov je obsežen in zajema velik razpon vrednosti, ki se lahko uporabijo. Malo je verjetno, da bi kdaj uporabili vse parametre pri izvedbi določenega zahtevka, vendar jih je dobro poznati[18]. Poleg pogosteje uporabljenih parametrov je potrebno omeniti še parameter *global*, ki omogoča (če vrednost *»true«*) oziroma onemogoča (če vrednost *»false«*) proženje, tako imenovanih, globalnih funkcij Ajax-a. Te se izvedejo v določenih fazah med procesiranjem zahteve. Glede na trenutno točko nahajanja zahtevka v njegovem ciklu izvajanja, se pokliče ustrezna funkcija. Ena od njih je funkcija *ajaxError()*. Izvrši se nad elementom z določenim identifikatorjem, znotraj katerega se prikaže sporočilo o napaki (28), če je napaka v vrnjenem odgovoru.

```
$('errorDiv').ajaxError(povratno_klicna_funkcija);
                                                                    (28)
```

Poleg tega imamo na razpolago še druge funkcije, ki ustrezajo ostalim stanjem, v katerih se lahko zahtevek nahaja in se izvršijo ob ustreznih dogodkih (Tabela 4.2).

Tip globalne funkcije Ajax	Faza izvedbe zahteve
ajaxStart()	Ob klicu funkcije *$.ajax()*, preden je primerek objekta *XHR* ustvarjen.
ajaxSend()	Ko je primerek objekta *XHR* ustvarjen, preden je zahteva poslana na strežnik.
ajaxSuccess()	Po vrnjenem zahtevku s strežnika in statusni kodi, ki označuje uspešno prejeti ogovor.
ajaxComplete()	Po uspešno vrnjenem zahtevku s strežnika in po izvedbi klica *ajaxSuccess()* ali *ajaxError()*.
ajaxStop()	Po zaključenem procesiranju celotnega zahtevka in po končani izvedbi vseh predhodno uporabljenih globalnih funkcij.

Tabela 4.2: Celoten seznam globalnih funkcij Ajax

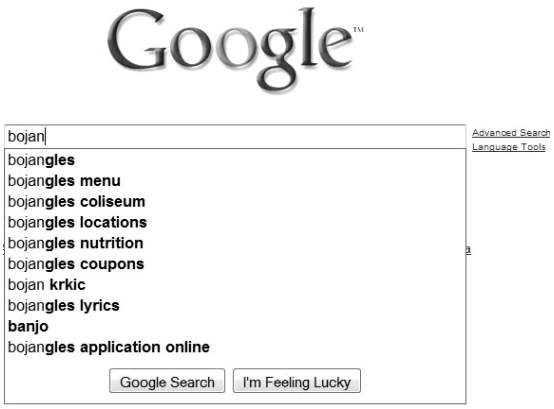
4.3.2 Samodejni dopolnjevalec besed – študija primera

Samodejni dopolnjevalec besed je namenjen dopolnjevanju besed ali besednih zvez, ki so predlagane na podlagi že vpisanih znakov uporabnika v za to namenjeno vnosno polje, ne da bi bilo pri tem potrebno dejansko vnesti besedo v celoti. Predlaganje možnih zadetkov se vrši

[18] http://docs.jquery.com/Ajax/jQuery.ajax#options

v realnem času, kar omogoča uporabniku izbiro vedno najbolj relevantnih predlogov iz prikazanega seznama. Tak način delovanja ne povzroči samo hitrejšega izvajanja aplikacije, ampak tudi poveča njeno učinkovitost.

Osnovni primer delovanja samodejnega dopolnjevalca lahko vidimo pri uporabi iskalnika *Google*[19] (Slika 9).



Slika 9: *Google Suggest.*

Vidimo, da se ob vnosu dela besede prikaže seznam najbolj relevantnih predlogov v padajočem meniju, iz katerega izberemo ustrezen predlog. Ti se z vsakokratnim novim dopolnjevanjem besede prikažejo na novo, tako da imamo vedno na voljo njihov najustreznejši nabor. S klikom na določen predlog iz seznama pa že sprožimo iskanje in pridobimo prikazane zadetke na podlagi izbranega predloga.

S stališča razvijalcev ni nujno komponenta samodejni dopolnjevalec besed uporabljena samo za namen spletnega iskanja, ampak je lahko njena uporaba namenjena tudi iskanju po lokalnih podatkovnih zbirkah. Skratka, uporabna je v vsakem spletnem projektu, kjer se izkaže potreba po hitrem predlaganju različnih možnosti uporabniku.

V našem primeru bomo za izvedbo komponente samodejnega dopolnjevalca besed uporabili orodjarno Mootools. Implementacija bo vsebovala tehniko Ajax ter še nekatere druge koncepte, predstavljene v tej nalogi (npr dedovanje, razrede).

Značilnosti komponente samodejnega dopolnjevalca besed

Orodjarna Mootools ponuja različne lastnosti, ki jih lahko uporabimo za izvedbo samodejnega dopolnjevalca besed. Zahtevnejša implementacija vsebuje koncepte dedovanja, razrede, tehniko Ajax. Vse to naredi komponento veliko bolj uporabno in prilagodljivo za različne spletne projekte. Obravnavana študija primera samodejnega dopolnjevalca besed uporablja vse že naštete koncepte, vendar bistven poudarek pri njeni implementaciji, zaradi uporabe tehnike Ajax, temelji na naslednjih treh točkah:

- asinhroni zahtevki se z vsakokratnim vpisom oziroma dopolnitvijo fraze pošiljajo na strežnik v rednih časovnih presledkih;
- če uporabnik ne vpiše ničesar od zadnje poslane zahteve, se nova zahteva ne pošlje;

[19] http://www.google.com/webhp?complete=1&hl=en

- pridobljeni podatki se ohranijo in so uporabni ob morebitni ponovitvi vnosa predhodne fraze. Na podlagi tega je možno prihraniti nekaj nepotrebno izvršenih zahtevkov na strežnik.

Poleg tega je izvedba komponente samodejnega dopolnjevalca besed razdeljena v več razredov. Osnovni razred (*Autocompleter)* služi določevanju osnovnih nastavitev in funkcionalnosti ter ga ne uporabljamo za namen tvorjenja novih objektov. Je podlaga za izdelavo podrazredov (*Autocompleter.Request, Autocompleter.Request.HTML, Autocompleter.Request.JSON, Autocompleter.Local*), ki dedujejo od njega in dodajajo še nekatere nove funkcionalnosti.

Razred Autocompleter

Razred *Autocompleter* je osnovni razred komponente samodejnega dopolnjevalca besed. Z njim določimo različne nastavitve, ki vplivajo na njegovo izvajanje in izvajanje njegovih podrazredov. Nekatere nastavitve običajno že vsebujejo privzete vrednosti, vendar jih ob tvorjenju novih primerkov objektov lahko ponovno določimo. Najpomembnejše nastavitve so [12]:

- ***minLength:*** *(celo število)* minimalna dolžina niza, ki jo mora uporabnik vnesti, preden so predlogi prikazani;
- ***markQuery****: (boolean)* ali označi del vsakega predloga, ki se ujema z uporabnikovim vnosom;
- ***width****: (mešano)* lahko je niz znakov, s katerim določimo širino padajočega menija s predlogi, ki je enako široko kot vnosno polje. Lahko je celo število, s katerim določimo poljubno širino vnosnega polja;
- ***maxChoices****: (celo število)* največje število predlaganih besed za prikaz. Privzeta vrednost je 10;
- ***injectChoice****: (funkcija)* ustvari vsako predlagano besedo iz seznama v padajočem meniju;
- ***customChoice****: (element)* z njih lahko določamo različne elemente *HTML*, v katere bodo vstavljene predlagane besede;
- ***delay****: (celo število)* čakalna doba pred prikazom padajočega menija s predlaganimi besedami oziroma čakalna doba pred njihovim posodabljanjem;
- ***selectMode****: (niz)* imamo tri različne načine uporabe predlogov:

  - *selection* – uporabnik lahko vpisuje želene vrednosti v vnosno polje, ne da bi se te na kakršenkoli način spremenile ali bile označene. Za uporabo ustrezne besede iz seznama predlaganih je potrebna izbira ene izmed njih;
  - *type-ahed* – ob začetem vnašanju uporabnika se v prikazanem seznamu predlaganih besed označi prva med njimi. Razlika med delom besede, ki jo je uporabnik vnesel, in predlagano besedo se dopolni z označenimi manjkajočimi črkami te besede. S pomikom kurzorja v desno smer sprejmemo predlagano

dopolnitev ali nadaljujemo s pisanjem in tako spremenimo predlagan del besede;

- *pick* – uporabnik lahko vpisuje želene vrednosti v vnosno polje, ne da bi se te na kakršenkoli način spremenile ali bile označene. Za izbiro ustrezne predlagane besede je potrebna uporaba smernih tipk (gor, dol). Pri tem se vnosna vrednost, glede na izbran predlog, ustrezno spremeni, vendar tu del besede, ki se spreminja, ni označen*;*

–***multiple****: (boolean)* ali se vnosna vrednost loči na več delov. Naslednji dve lastnosti vplivata na obnašanje te, če je njena vrednost nastavljena na *»true«*;

–***separator****: (string)* ločevalec, ki določa, kako bodo besede ločene med seboj (npr. ', ' vejica);

–***separatorSplit:*** *(ReqExsp)* regularni izraz, na podlagi katerega ločimo vnosne vrednosti;

–***cache****: (boolean)* ali se hrani vrednosti, ki so bile že prikazane.

Možnost določevanja velikega števila nastavitev omogoča večjo mero fleksibilnosti in prilagajanja različnim potrebam. Še več dodatnih specifičnih nastavitev pa določamo na začetku znotraj vsakega podrazreda.

Temu sledi tako imenovani inicializacijski del, v katerem določimo vhodne argumente (29):

–***element:*** je referenca na element, na katerem bo izvedena komponenta samodejnega dopolnjevalca besed (običajno tekstovno polje);

–***vir podatkov:*** podatke lahko pridobimo na lokalni ravni (iz lokalne tabele) ali s pomočjo tehnike Ajax iz oddaljene lokacije (strežniška skripta). Argument vir podatkov določimo samo v podrazredih;

–***nastavitve:*** določimo tiste nastavitve, ki še niso določene kot privzete znotraj razreda oziroma jih ponovno določimo, če želimo njihove vrednosti prepisati.

```
initialize: function(element, options) {
        this.element = $(element);
        this.setOptions(options);
        this.build();
        this.observer = new Observer(this.element,
                this.prefetch.bind(this), $merge({
                        'delay': this.options.delay
        }, this.options.observerOptions));
        this.queryValue = null;
        if (this.options.filter) this.filter =
                this.options.filter.bind(this);
        var mode = this.options.selectMode;
        this.typeAhead = (mode == 'type-ahead');
        this.selectMode = (mode === true) ? 'selection' : mode;
        this.cached = [];
},
...
```
(29)

V tem delu (29) določimo glavne metode, ki se morajo najprej izvršiti. V našem primeru je to metoda *build().* Služi izgradnji strukture *HTML*, ki je osnova za prikazovanje predlaganih besed, in dodajanju dogodkov elementom tej strukturi. Tu imamo kreiranje še enega primerka razreda z imenom *Observer*. Namenjen je zaznavanju sprememb v elementih obrazca, v našem primeru vnosnega polja, v katerega vnašamo besede. V preostanku inicializacije lahko določimo še dodatne spremenljivke, uporabljamo jih v celotnem razredu in se nanje sklicujemo s pomočjo besede *this*. Po zaključenem inicializacijskem delu so v razredu vsebovane še druge različne podporne metode, ki omogočajo izvedbo potrebnih funkcionalnosti. Nekatere izmed njih so določene, kot tako imenovani dogodki, izvedeni *»po meri«*. Gre za dogodke, ki jih prožimo sami in niso del standardnih dogodkov (npr. kot *onClick, onMouseOver, itd.*). Eden izmed teh dogodkov, uporabljenih v tem razredu, je dogodek *onSelect()*. Sproži se, ko uporabnik z miško ali smernimi tipkami premakne izbran del določenega predloga, ki bi ga rad spremenil.

Ta razred služi kot osnova razredoma, ki sta prav tako del komponente samodejnega dopolnjevalca besed. To sta razreda *Autocmpleter.Request* in *Autocompleter.Local.* Razlikujeta se glede na uporabljen vir, iz katerega pridobivata podatke za ustrezno predlagane besede. Prvi uporablja oddaljen vir (s pomočjo tehnike Ajax se vršijo asinhroni zahtevki na strežniško stran), drugi pa lokalnega (tabela vrednosti na strani odjemalca). Poglejmo si delovanje razreda *Autocompleter.Rquest*.

Razred Autocompleter.Request

Ta razred služi kot osnova za izvedbo tehnike Ajax in vsebuje še dodatne lastnosti in nastavitve, ki so potrebne pri njeni izvedbi, in so nadgradnja osnovnega razreda. Sprejme tri argumente [12]:

- ***element:*** *(string ali element)* niz znakov, s katerim identificiramo element ali referenco na element;
- ***url:*** *(niz)* spletni naslov, na katerem pridobimo vrednosti za prikaz predlogov;
- ***options:*** (objekt) nastavitve določimo v objektu v obliki ime/vrednost;

Poleg tega lahko določimo še dodatne nastavitve. Te so [12]:

- ***postVar:*** *(niz)* ime spremenljivke za pošiljanje niza, ki ga uporabnik vpiše v vnosno polje;
- ***postData:*** *(objekt)* dodatni podatki, ki jih pošljemo v zahtevi;
- ***ajaxOptions:*** *(objekt)* dodatne nastavitve za razred *Request*[20];

Iz razreda *Autocompleter.Request* lahko razširimo še dva specifična razreda, iz katerih ustvarimo primerke razredov za pošiljanje asinhronih zahtevkov za to komponento:

- *Autocompleter.Request.HTML* (podpora formata *HTML*)

```
Autocompleter.Request.HTML = new Class({
```

[20] http://mootools.net/docs/core/Request/Request

```
     Extends: Autocompleter.Request,

     initialize: function(el, url, options) {
          this.parent(el, options);
          this.request = new Request.HTML($merge({
               'url': url,
               'link': 'cancel',
               'update': this.choices
          }, this.options.ajaxOptions)).addEvent('onComplete',
     this.queryResponse.bind(this));
     },
    ...
                                                                    (30)
```

–*Autocompleter.Request.JSON* (podpora formata *JSON*)

```
     Autocompleter.Request.JSON = new Class({

     Extends: Autocompleter.Request,

     initialize: function(el, url, options) {
          this.parent(el, options);
          this.request = new Request.JSON($merge({
               'url': url,
               'link': 'cancel'
          }, this.options.ajaxOptions)).addEvent('onComplete',
          this.queryResponse.bind(this));
     },
     ...
                                                                    (31)
```

Oba razreda vsebujeta še dodatne lastnosti, ki jih nastavimo s pomočjo klica funkcije konstruktorja *this.parent()* njihovega nadrejenega razreda. Poleg tega določa programska logika (30) še funkcionalnost Ajax in funkcijo *this.queryResponse()*, ki se bo vršila ob vsakokratni uspešno zaključeni zahtevi. Programska logika (31) se ne razlikuje veliko s predhodnim primerom (30). Razlika je le v tem, da gre za pridobivanje podatkov v obliki *JSON*. V tem primeru je potrebno poskrbeti, da strežniška stran vedno vrne podatke v ustrezno formatirani obliki *JSON* (npr. za jezik *PHP* bi bilo to: *echo json_encode($data);*).

S tako določenima razredoma lahko ustvarimo primerka obeh razredov (32, 33). V ta namen uporabimo operator *new* in določimo ustrezne argumente glede na razred, ki ga želimo uporabiti. Sintaksa za klic konstruktorja komponente samodejnega dopolnjevalca besed in kreiranje primerka objekta razreda *Autocompleter.Request.HTML* je sledeča:

```
//sintaksa
new Autocompleter.Request.HTML(Element_ID, Vir_URL, [nastavitve]);

//primer uporabe
new Autocompleter.Request.HTML($(inputWord), 'server/script.php', {
            'indicatorClass': 'autocompleter-loading',
            'postData': {
                    'extended': '1' //pošiljanje dodatnih podatkov
```

```
            },
            'injectChoice': function(el) {
                    //obravnava vrnjenih podatkov
                    var value = el.getFirst().get('html');
                    el.inputValue = value;
                    this.addChoiceEvents(el).getFirst().set('html',
                    this.markQueryValue(value));
            }
    });
```
(32)

Vidimo lahko, da poleg osnovnih argumentov določimo tudi funkcijo, ki skrbi za pravilen prikaz vrnjenih predlogov.

Če želimo podatke pošiljati v obliki *JSON*, je sintaksa za klic konstruktorja komponente samodejnega dopolnjevalca besed in ustvarjanje primerka razreda *Autocompleter.Request.JSON* sledeča:

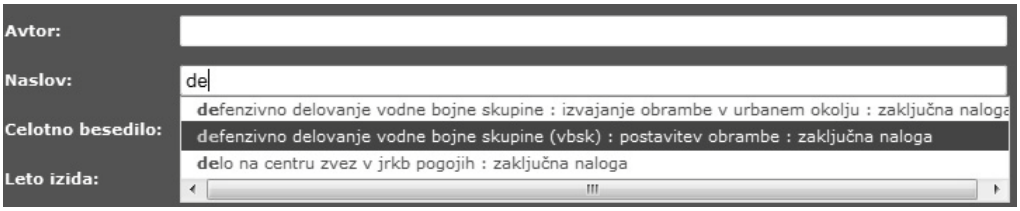
```
//sintaksa
new Autocompleter.Request.JSON(Element_ID, Vir_URL, [nastavitve]);


new Autocompleter.Request.JSON($(inputWord), 'server/script.php', {
      //določevanje nekaterih nastavitev
      'indicator': indicator,
      'multiple': true,
      'selectFirst': true,
      'selectMode': false,
      'minLength': 2
 });
```
(33)

V obeh premeri je potrebno določiti element *HTML*, za katerega se bo uporabila komponenta samodejnega dopolnjevalca besed. Enostavna struktura *HTML*, z vsebovanim elementom <*input*>, bi lahko izgledala tako:

```
    <form action="url_naslov" method="get">
    <label>
            <span>samodejni dopolnjevalec besedspan>
            <input type="text" name="beseda" id="inputWord" />
    </label>
    </form>
```
(34)

Pri določevanju take strukture (34) je potrebno biti vedno pozoren, da določimo ustrezen identifikator elementa <*input*>. Ta mora biti enak prvemu argumentu pri obeh klicih konstruktorja, če želimo, da bo delovanje komponente pravilno.

V obeh primerih ustvarimo komponento samodejnega dopolnjevalca besed, ki izgledata na zunaj povsem enako. Razlikujeta pa se v različnih nastavitvah in v obliki prenosa podatkov. V primeru pravilne izvedbe komponente bi moral biti rezultat podoben (Slika 10).



Slika 10: Samodejni dopolnjevalec besed.

5 SKLEPNE UGOTOVITVE

Rešitve v diplomski nalogi predstavljajo le delček tehnologij za izdelavo obogatenih spletnih aplikacij, ki so prisotne na trgu in jih lahko uporabimo v ta namen. Njihov pester izbor lahko še dodatno oteži celoten proces izdelave spletnih aplikaciji. Ta v začetku zahteva določitev ustrezne tehnologije, ki jo je potrebno dobro preučiti, saj lahko napačna ocena o primernosti tehnologije odločilno vpliva na potek izvedbe celotnega projekta. Torej, v začetku je smiselno vložiti dovolj časa in se poglobiti v to, kaj posamezne tehnologije omogočajo, in določiti njihovo primernost za uporabo.

Če se pri spletnem projektu odločamo, ali bi bilo primerno uporabiti nekatere naprednejše tehnologije, ki se v današnjem času uveljavljajo pri izdelavi obogatenih spletnih aplikacij, ocenjujem, da si s pomočjo te naloge lahko ustvarimo popolnejšo sliko, kaj predstavljene tehnologije ponujajo. S tem bi lahko olajšali odločitev o izbiri platforme, na kateri bi temeljila izgradnja spletne rešitve. Pri izbiri ustreznih platform se je težava pokazala v tem, da je bilo potrebno primerjavo omejiti in tako izpustiti nekatere platforme, ki se poskušajo uveljaviti na področju razvoja obogatenih spletnih aplikacij. Mislim, da so v tej nalogi zajete bolj uporabljene.

Izpostavitev platforme spletnega brskalnika med vsemi ostalimi se morda zdi komu neprimerna. Vendar imajo pri tem veliko vlogo pristop Ajax in orodjarne Javascript, v povezavi z odprto-kodnimi strežniškimi tehnologijami (npr. *PHP, Ruby, Python, Java*). Te so, v primerjavi z Adobovimi (*Flex*) in Microsoftomi (*Silverlight*) rešitvami, v prednosti, saj za potrebe razvoja ne zahtevajo nakupa licenčnih orodji in so tako dostopnejše posameznikom in manjšim podjetjem. Zato je bilo vodilo izpostaviti platformo spletnega brskalnika in predstaviti tehnologije, ki so dostopne vsakemu razvijalcu. Osnovna ideja predstavitve tehnike Ajax je v tem, da se seznanimo, kaj je Ajax, čemu je namenjen in kako deluje. Na ta način lahko vsak, ki se prvič sreča s to tehniko, pridobi osnovno znanje o njej.

Če želimo spletne aplikacije izvesti s pomočjo tehnike Ajax in nočemo uporabiti orodij, ki ponujajo celotno razvojno integrirano okolje, je najprimernejša uporaba orodjarn Javascript. Z njimi nadgradimo osnovno znanje tehnike Ajax in odpravimo njene mnoge pomanjkljivosti, ki izvirajo iz uporabe osnovnega jezika Javascript. Zato sem se v tej nalogi osredotočil na primerjavo dveh, med razvijalci bolj priljubljenih orodjarn Javascript. Bolj so poudarjene konceptualne kot tehnične razlike, ki se pojavljajo med njima obema. Slednje so uporabljene samo toliko, da omogočajo ustrezno predstavitev konceptualnih razlik. Njihovo poznavanje lahko marsikateremu razvijalcu olajša odločitev o pravilni izbiri orodjarne oziroma o primernosti orodjarne za izvedbo projekta, ki ga želi realizirati. To je res samo v primeru, če gre za odločitev med orodjarnama, ki sta opisani v tej nalogi. Težava pri tej primerjavi je bila, kateri dve orodjarni vključiti v primerjavo. Glede na to, da obstajajo še druge orodjarne, ki so prav tako dobre rešitve, je bila odločitev kompromisna. Orodjarno Mootools sem vključil v primerjavo, ker sem jo največ uporabljal, orodjarno Jquery pa zato, ker je med razvijalci najbolj razširjena.

Kot nadgradnjo te naloge bi lahko raziskali in primerjali uporabo orodjarn, ki so bolj povezane s strežniškimi tehnologijami oziroma so že integrirane vanje. Zajemala bi lahko rešitev, ki jo ponujata podjetji *Google (s produktom GWT)* in Zend Tehnologies (*s produktom Zend framework*). Slednji je v svoje ogrodje *Zend Framework*, ki je namenjeno razvoju s pomočjo jezika *PHP*, integriral delovanje orodjarne Javascript Dojo.

Menim, da bodo tehnologije za razvoj obogatenih spletnih aplikacij v prihodnosti napredovale tako, da bodo postale še prijaznejše do uporabnikov. Med temi, ki jih danes uporabljamo, velja izkoristiti potencial orodjarn Javascript, saj z njimi lahko izvedemo večino tega, kar naj bi danes vsebovale tehnološko napredne spletne aplikacije.

6 PRILOGE

Zgoščenka z izvorno kodo študija primera.

7 PONAZORILA

SLIKE

TABELE

8 LITERATURA

[01] Allaire J., Macromedia Flash MX – A next-generation rich client. Dostopno na: http://www.adobe.com/devnet/flash/whitepapers/richclient.pdf.

[02] Brijesch D., Rich internet applications: Look into available technology choice. Dostopno na: http://www.jaxmag.com/itr/online_artikel/psecom,id,828,nodeid,147.html.

[03] Cascading style sheets, Dostopno na: http://en.wikipedia.org/wiki/Cascading_Style_Sheets.

[04] Chaffer J., Swedberg K., Learning jQuery 1.3, Better interaction design and web development with simple Javascript tehniques, Birmingham, Packt publishing, 2009, pogl 6.

[05] Deitel P. J., Dietel H. M., Ajax, rich internet applications, and web development for programmers, Person Education, 2008, str 5,6.

[06] Garrett, Jesse J., Ajax: A New Approach to Web Applications. Adaptive Path, http://www.adaptivepath.com/publications/essays/archives/000385.php.

[07] Helvg S. Nada T., »Rich internet application: Tehnical comparisson and case studies of Ajax, Flash, Java based RIA«, UWEBC Opinion Paper, 2005.

[16] Johnson D., White A., Sharland A., Enterprise Ajax, Strategies for building high performance Web applications, Prentice Hall, 2007, str. 100 - 106.

[09] Katz Y., Bibeault B., jQuery in action, Manning, 2008, pogl. 8.

[10] Mahemoff M. Ajax Design Patterns, O'Reilly, 2006, pogl. 1.

[11] Miscrosoft Silverlight. Dostopno na: http://www.microsoft.com/silverlight/.

[12] Mootools autocompleter component. Dostopno na http://www.clientcide.com/docs/3rdParty/Autocompleter.

[13] Msdn Silverlight developer center. Dostopno na: http://msdn.microsoft.com/en-us/library/bb404700(VS.95).aspx#WhatFeatures

[14] Newton A., Mootools essentials: The official Mootools Reference for Javascript and Ajax development, Apress, 2008, pogl. 10.

[15] O'Reilly Media Inc., Mastering Ajax, Advanced requests and responses in Ajax, 2006 http://www.ibm.com/developerworks/web/library/wa-ajaxintro3/index.html?S_TACT=105AGX08&S_CMP=EDU.

[16] Rich client platform. Dostopno na: http://en.wikipedia.org/wiki/Rich_Client_Platform.

[17] Ramirez F., Wroblewski L., Web applications solutions: a designers's guide. Dostopno na: www.lukew.com/resources/WebApplicationSolutions.pdf.

[18] Steele O., Web Mvc, Dostopno na: http://osteele.com/archives/2004/08/web-mvc.

[19] Thin client. Dostopno na: http://en.wikipedia.org/wiki/Thin_client#Application_program.

[20] Valdes R., Hype cycle for web and user interaction Technologies: RIA platforms, Gartner, 2008, str. 24.

[21] Valdes R, Snippets, widgets and other levels of AJAX development, 2006, Dostopno na: http://adtmag.com/Articles/2006/03/01/Snippets-widgets-and-other-levels-of-AJAX-development.aspx.

[22] Zetie C., The rise of rich internet applications. Dostopno na: http://whitepapers.zdnet.com/abstract.aspx?kw=the+rise+of+rich+internet+applications&docid=174760